

Karl Moore's Visual Basic .NET: The Tutorials

KARL MOORE

Apress™

Karl Moore's Visual Basic .NET: The Tutorials
Copyright © 2002 by Karl Moore

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-021-X

Printed and bound in the United States of America 12345678910
Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

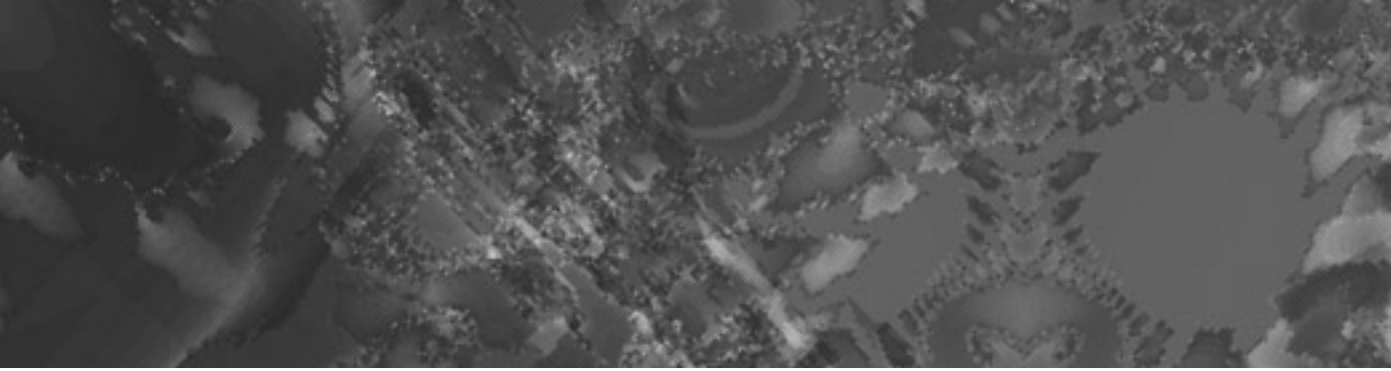
Technical Reviewers: Patricia Moore, Richard Costall, Ray Ellison, Erik Giggey, Donald Carter
Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Karen Watterson, John Zukowski
Managing Editor: Grace Wong
Project Manager: Alexa Stuart
Copy Editor: Tom Gillen
Production Editors: Janet Vail and Grace Wong
Composer: Impressions Book and Journal Services, Inc.
Indexer: Ann Rogers
Cover Designer: Tom Debolski
Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010
and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.
In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit <http://www.springer-ny.com>.
Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 Ninth St., Suite 219, Berkeley, CA 94710.
Phone: 510-549-5903, Fax: 510-549-5933, Email: info@apress.com, Web site: <http://www.apress.com>

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.



TUTORIAL 6

Services Rendered

Introducing the World of Web Services

“Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.”—Richard Cook

TRUTH BE KNOWN, COMPUTERS ARE highly unsociable creatures.

I mean, they won't even talk to each other unless you string a mass of cable between them. And, if you're actually looking to get them engaged in any sort of productive conversation, you'll need to tweak network settings, twist hub knobs, pray nightly to the almighty power of Bill Gates, *and* cross your fingers.

Thankfully, however, there's a cure for such typically timid devices. It's the .NET Framework, and it works like alcohol for computer chips. Install this baby and suddenly your quiet chunk of silicon seems better at schmoozing than Dale Carnegie.

Why? Because that .NET Framework includes ASP.NET. And ASP.NET includes Web services. And Web services give you the ability to write programs on different machines that can talk to each other, easily.

Your client program might ask your main computer for information about a customer, for example. Or query the government for its latest tax rates, say. Or nab late-breaking headlines straight from CNN. Or *whatever*.

It's all possible, when you get talking—via Web services.

Could be useful? Fancy learning more? Then sign up and step this way . . .

How It All Works

Sometimes you simply can't do everything on one computer. It's often best to separate the functionality, using programs that talk to and help applications on different computers. Enter stage left: *Web services*.

Now, imagine you work for Deals on Wheels, the most popular car dealership this side of the Mississippi. The whole group runs on your applications, and, with five popular sites, you're exceptionally pleased.

The problem is your users aren't. And, in stereotypical user fashion, they do nothing but moan and request new features. (Difficult to imagine, I know.) Anyway, first on their list of improvements is an automatic credit check performed on each client as soon as their details are entered into your system.

Oh, great. I mean, how on Earth can you implement something like that? Hmm, I guess you could just write the credit-checking code yourself, then update all the users with a new version of your customer database program. But surely that wouldn't be quite so comprehensive as a third-party check, and what about that "credit check free" day they were talking about? Yuck. What an administrative nightmare.

Actually, maybe it's just easier to tell the users that they should physically call your checking agency. After all, *they've* got the information, and *your* computers can't talk to *theirs*, can they?

What's that? Oh, your boss Dodgy Del has his own personal request, too? When a customer steps through his doors to trade in a rusty old motor, Sam the Salesman instantly heads to his desk and checks out the trade-in value in his pricing guide. Problem is, this is often the 1986 pricing guide—and Del guesses he's losing thousands of dollars a month due to out-of-date information.

He suggests a centralized database containing all the latest prices—and you shiver, thinking of all the different applications you'll need to update for this. Reams of data access code in three different languages flash before your eyes. But, after all, there's no easier way, is there?

Hmm, I could just say "no" to both of these questions, but then that'd make for a real short chapter. Instead, I'm going to answer "yes," and welcome you to the world of Web services, part of ASP.NET. Erm, so just what are Web services?

Web services are a way of creating programs that talk to and help applications on different computers. You have a server machine that "exposes" the service and clients that "discover" and "consume" that service.

So, your credit-checking agency may make a service available over the Internet that allows you to pass details of your customer to it in code, and the service will return the credit rating. Or you might create your own service on the main company server that looks up car prices in a database, and then simply call this one service from each of your applications. That's what Web services allow you to do: get programs on different computers talking together. They allow you to *distribute* the functionality of your application. How could something like this help you, right now?

Hmm, you still look a little suspicious. I'm guessing you think I've slipped off into my own little fantasy world once again and that this sounds all too good to be true. Well, I've had my cold shower, and can inform you that it *is* possible, right now, today, with Web services.

How? Well, you (or perhaps that credit-checking agency) start by creating a Web service project in Visual Studio .NET. This is basically just a project containing code, a little like the class library we worked with in Tutorial 5, "Using Objects". Then the development begins, and our friendly coder tells VB .NET to show certain bits of his code to the world.

So, maybe he'll tell it to expose that DoCreditCheck method or perhaps that GetCarPrice function.

When finished, the code is compiled, and that "Web service" is made available on a master computer somewhere—our network, the Internet, or wherever. And that's it for your Web service: it's up, running, and waiting to serve.

Next, it's time to develop your client, the program that will use the Web service. You start by "discovering" the Web service and its features, then use all those exposed functions exactly as you would a local piece of code, and they run, all the way over on the other machine. It all just works: you do nothing special.

The Geeky Bit

That's right: *you* do nothing special. However, our lovely .NET Framework is being smarter than da Vinci in Gucci. When you run the functions belonging to that Web service, it calls the remote machine passing any parameters, runs the Web service code, and returns any data back to your client program *automatically*. Complete doddle.

And all of this works via something called SOAP. Shower jokes aside, *SOAP* stands for the *Simple Object Access Protocol* and refers to how Web services talk. And how do they talk? Behind the scenes, everything is converted and passed about as XML, which is just plain, structured text. And this XML is transferred over HTTP, which is typically used for viewing Web pages. (See Figure 1-1.)

Why is this clever? Two reasons. First off, XML is pure text, and pure text is *speedy* and *understood by all platforms*. And, because it works via HTTP, it also bypasses any annoying firewalls that may attempt to thwart your development efforts. But, again, this is all handled automatically for you, so there's no need for concern. Again, it's just a complete doddle.

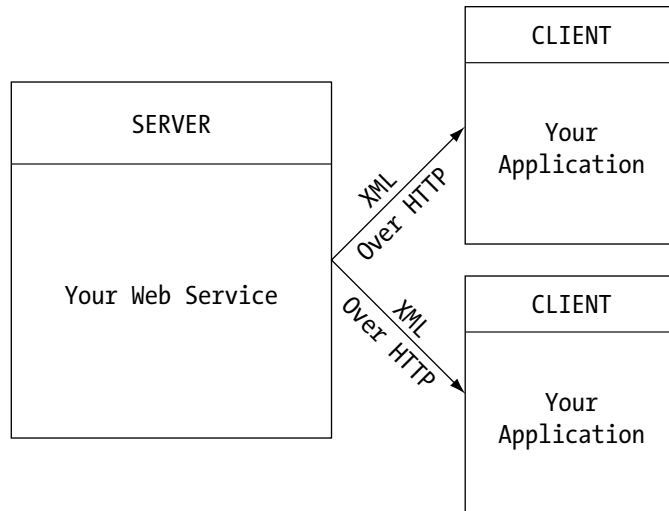


Figure 1-1. A Web service in action

And if you're coming from the old school of DCOM and all that jazz, you're in for a particularly pleasant surprise: with Web services there's no registering, no settings to fine-tune, no annoying tight-coupling. Basically, it's a complete doddle. Did I mention that?

So, Web services allow you to get computers talking. You expose certain functionality through a service, and your clients “discover” and “consume” that service, live via XML over HTTP.

Ready to start exploring Web services in real life? Let's dive into meat space and put all this fantasy into practice. Oooh, matron!

Creating a Web Service

George Bush (Sr.) once said, “I pushed the button down here and one up here with the green thing on it. And out came a command to somebody that I had written.”

Uh-huh. Well, perhaps creating Web services wasn't for him. But, for us, it's time to put all this theory into action, getting our palms dirty with a little hands-on.

1. Launch Visual Studio .NET and create a new ASP.NET Web service, changing its location to `http://localhost/MyHelper`.

After a little whizzing and whirring, you should be presented with a pretty dull-looking screen. This is `Service1.aspx.vb`, the code file behind your actual service. However, I don't see any code right now, presumably because we're in design mode. Let's change that.

2. On the menu, select View > Code to open the code window.

TOP TIP *Instead of constantly using the menu to open the code window, just press F7. It'll take you straight to the code behind the object you're working with.*

You should see a small mound of code in front of you, although nothing shockingly horrid. We have a line that imports the Web service namespace, making its functionality available to use. Aside from that, it just looks like a regular class that inherits from the WebService class. It also includes a bundle of comments, currently wearing a very chic shade of green.

Now, let's add a function to this class—just a bog-standard, run-of-the-mill function.

3. Add the following function to your class:

```
Public Function Reverse(ByVal Text As String) As String
    Return StrReverse(Text)
End Function
```

Well, it beats a simple Hello World sample, but only *just*. Yes, it's a chunk of code that takes a string and reverses it, but, unless you're about as bright as a blown bulb, I guess you already figured that out.

Anyway, now it's time to reveal the real secret of Web services. It's the key to instantly exposing your data to the world, it's the backbone to the entire .NET concept, it's . . . <WebMethod()>.

Yes, I know. Profound. But just by adding this simple “attribute,” we're telling VB .NET that this is a “WebMethod,” a method or function that should be made available via your network, the Internet, or wherever.

4. Add the <WebMethod()> attribute to the beginning of your function, so it reads:

```
<WebMethod()> Public Function Reverse(ByVal Text As String) As String
    Return StrReverse(Text)
End Function
```

And surprise, surprise—that's our Web service finished! Let's build our solution, then see what this simple <WebMethod()> attribute has done for us.

- From the menu, select Build > Build Solution (or press Ctrl+Shift+B).

Accessing the Web Service from a Browser

There are two key ways to access your Web service. The first is from a Web browser. That's our cue.

- In your application, press F5 to start your Web service.

Your default browser should fire up, directing itself to somewhere like `http://localhost/MyHelper/Service1.aspx`.

What you're looking at in Figure 1-2 is an automatically generated interface to your HTTP Web service.

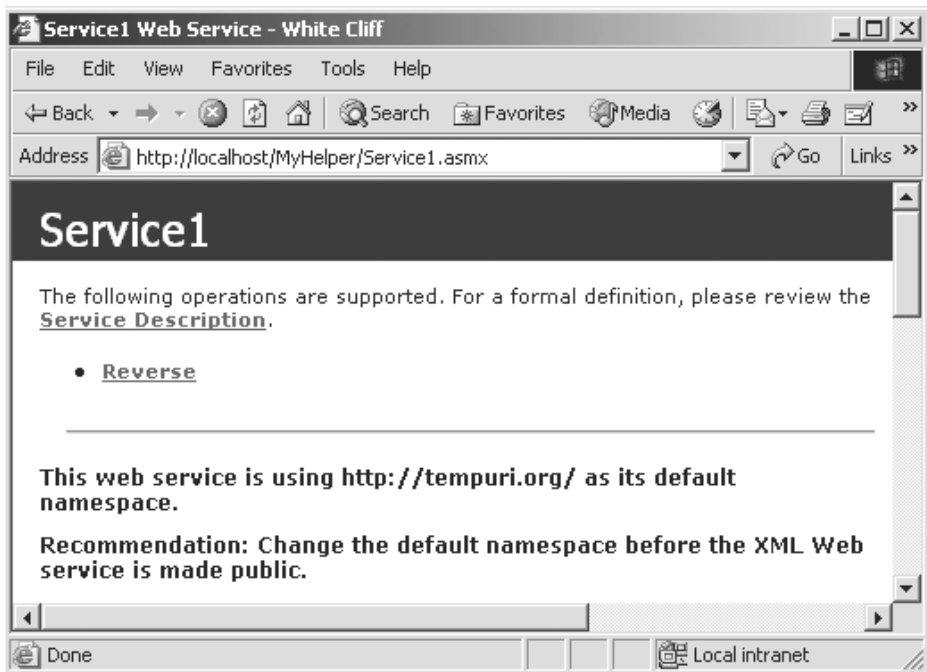


Figure 1-2. Accessing our Service from a browser

Here, we can see our Web service is very originally entitled “Service1” and that it exposes one operation called “Reverse”.

- Click on the Reverse link.

You'll be taken to a page that describes our Reverse operation. From here, you can either test it directly or check out some simply *thrilling* sample SOAP code. Think I'll just test it.

3. Enter a value for the Text parameter, such as your name.
4. Click on the Invoke button to run your function.

A separate window should pop open and display something like:

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">erooM IraK</string>
```

Huh? Well, this is XML, that “eXtensible Markup Language” we talked about earlier and stumbled across back in Tutorial 2, “Doing Databases”. Here, this chunk of XML is telling us that our Reverse operation has returned a string saying “erooM IraK”. Think I’ll just stick to *Karl*.

TOP TIP *XML is the glue that holds a lot of .NET together. It's essentially a self-describing structure that can hold hierarchical data. Or at least that's the official lowdown. Personally, I just think, "Hmm, looks a bit like HTML." Simple minded, you see.*

5. Close your browser windows and click on the Stop button in VB .NET.

So, that's a Web service. Okay, admit it: it looks about as useful as an ice machine at the North Pole. But this is only the raw Web interface. There's another, much more powerful way of accessing them, a way that completely hides all the XML from us and makes development easy.

I'm talking about using Web services . . . in your applications.

Accessing the Web Service from an Application

Next, let's throw together an application that actually “consumes” this Web service. You can create this on the same machine as your Web service if you like, or on another computer in your network. It really doesn't matter; as long as your browser can access the Web service, your program will be able to also.

1. Create a new Windows application using Visual Studio .NET.
2. From the menu, select Project > Add Web Reference.

You should be looking at the Add Web Reference screen. If not, either your machine has contracted the Squiggle virus, or you've accidentally unplugged your monitor. Next, you need to tell VS .NET where your Web service is located.

3. Type in your Web service ASMX address, which will be something like `http://localhost/MyHelper/Service1.asmx`, and press the Enter key.

TOP TIP *If your Web service is on the local machine, you can try clicking on the “Web References on Local Web Server” link to locate your service. If your service is on another machine in the network, replace the “localhost” portion of the sample address with the actual machine name, IP address, or Web address—basically, whatever you'd type into a Web browser to access your Web service.*

4. When your Web service appears, click the Add Reference button. (See Figure 1-3.)

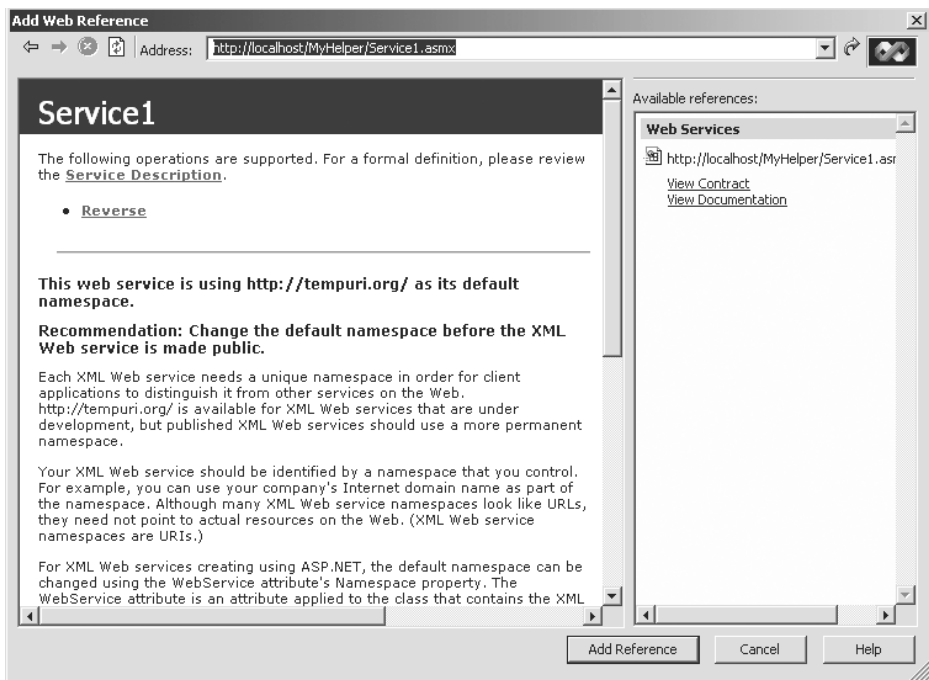


Figure 1-3. Adding a reference to our Web service

Take a glance at the Solution Explorer. You'll notice a new Web References entry there, with "localhost" or your machine name just underneath. Try expanding this node, browsing all the files underneath it. These have all been automatically generated and exist to tell VB .NET how to talk to the service on that machine.

5. Draw a button out onto the Form designer.
6. Behind the click event of the button, add the following code:

```
Dim objHelper As New localhost.Service1()
```

Here, we're treating our Web service just like any regular object we've seen so far in this book. Once again, you may need to change "localhost" to your machine name, as listed in the Solution Explorer.

TOP TIP *Our code here is actually working with a "helper" class that Visual Studio .NET automatically creates for you. When you use its properties and such, this "wrapper" class makes the Web service call, processes the results, and passes them back to your code. If you want to look at the Helper class code, click on "Show All Files" in the Solution Explorer, then expand the Web References folder, then localhost. For each Web reference here, you'll see two XML-based files: a DISCO file that provides details on discovering the service, and a WSDL (Web Service Discovery Language) file that provides details on the capabilities of the service. Expand that ServiceName.WSDL file and look at the ServiceName.VB file underneath it. This is the Helper class code. You can even edit it, if you're feeling particularly randy.*

That's our new objHelper object instantiated. Now we're ready to use it. Try typing objHelper and pressing the period key immediately after. What happens? (See Figure 1-4.)

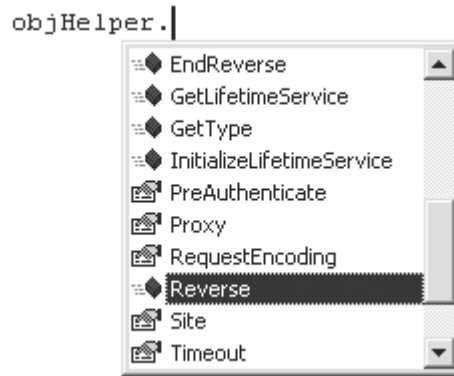


Figure 1-4. Our Web service in the code window

What's that? You don't remember coding all those features? Well, you're either a Steve Jobs reincarnate who can program subconsciously (and at rather an impressive pace), or these are simply standard extras above and beyond your own simple function, extra widgets automatically added to help you use the Web service. I might be wrong, but I'd go for option two.

Let's use our Reverse function in code now. Here, we're simply going to display a message box, telling VB to display the result of Reverse on "Karl Moore". (That's me, by the way.)

7. Underneath your existing button click code, add the following code:

```
MessageBox.Show(objHelper.Reverse("Karl Moore"))
objHelper = Nothing
```

Understand? We're just telling it to run our function with this particular string and to display the results in a message box. Finally, we just set our objHelper variable to nothing, telling it that we're no longer using it: we're just being tidy, as learned how back in Tutorial 5, "Using Objects".

8. Press F5 to test your program.
9. Click on your button.

What happens? After a short delay, you should get back “erooM IraK”.

Now try clicking on the button a second time. See what happens? First time round it was about as slow as a slug. To be specific, a dead slug. Now it zaps away at the speed of light, having already made that initial connection. Just a consideration.

TOP TIP *Wondering how to discover and consume a Web service in something other than a Windows application? Perhaps you want to use the functionality from an ASP.NET Web application or maybe even another Web service? No worries: it all works in exactly the same way. Just “Add Web Reference” and start coding. There’s no difference.*

If you’ve got a few minutes before moving on, try creating a function that returns an integer as opposed to a string. How does the XML differ? Also, can you use multiple parameters just as you can with a regular function? One more test: try adding a method or function that doesn’t include that WebMethod attribute. What happens? Can you see it via your browser?

Getting bored yet? No? Oh, well, let me tell you about my Victorian stamp collection. Oh, you *are* getting bored? Right then, let’s move on and imagine some of the possibilities here.

Imagining the Possibilities

What’s that? The complexity of your organization slightly surpasses our Reverse sample? Oh, darn. Sorry about that. Maybe I can offer you a coupon against my next book or something.

But, you see, although we’ve covered only the core essentials here—and pretty quickly, too—it’s still more than enough for you to get your own Web services into production. It’s all about <WebMethod(>.

Just imagine the possibilities open to you, right now. Imagine looking up car trade-in prices from any of your applications, simply by using your one centralized function. Imagine adding new customers just by calling an Add method on your Customer Web service. Imagine being able to change any part of your application code just by editing one project on your server.

<takes deep breath>

Imagine consuming a third-party, Internet-based Web service to authorize a MasterCard transaction or to check a customer’s credit. Imagine exposing your own product stock information over the Web, or making data such as weather reports and sports scores available to your service subscribers.

TOP TIP *The UDDI Directory from Microsoft attempts to list many of the third-party Web services available for you to consume. You can view the directory by visiting <http://uddi.microsoft.com/visualstudio> in the Add Web Reference dialog box. Alternatively, search it by clicking the XML Web services link on the Visual Studio .NET start page. Another useful directory can be found at <http://www.xmethods.com>.*

At the time of going to press, among the public Web services in action were a live dictionary and thesaurus, a horoscope generator, an SMS service, an NFC headlines application, and a music teacher search engine. Diverse? Oh yes. *Just imagine.*

And you think that's all? No, sir. You see, with all these possibilities, passing about plain data is often not enough. You need to crank up the volume, with *smart data*—and we'll see at how you can do that, next.

Passing Smart Data

The problem with using Web services as we've shown here is that they're a little limiting. I mean, let's say that you've created a groovy customer management Web service on your network. Perhaps it contains an Add method that accepts details of your customer and adds them to the database. No problem. But what about actually *retrieving* your customers?

Think about it. Functions can return only one item, such as a True or False, or perhaps a string. They can't, for example, return a list of customers and their orders . . . unless, of course, you pass around *smart data*.

I'm talking about relational, multitable DataSets. I'm talking about arrays. I'm talking about classes. I'm talking about structures. I'm talking about smart data, data containing multiple portions that can be passed back through a function—even in a Web service.

To demonstrate the concept, I'd like to create a Web service that opens a database of your choice, then passes all the required data from one particular table data back to your user as an easy-to-use, typed DataSet. Okay, let's get started.

1. Create a new ASP.NET Web service at <http://localhost/MyDataService>.

Now, we're going to start off by using the Server Explorer to connect to your database. I'm going to leave this bit up to you; you can connect to your company database, one of the sample Access or SQL Server databases we created back in Tutorial 2, "Doing Databases", or perhaps just infamous Northwind. Your call.

Personally, I'm going to connect into the Surgery.mdb sample we created back in Tutorial 2, "Doing Databases".

2. From the menu, select View > Server Explorer.
3. If your database is not shown under Data Connections or as an SQL Server database under the Servers entry, click on the "Connect To Database" button and enter the details. (See Figure 1-5.)

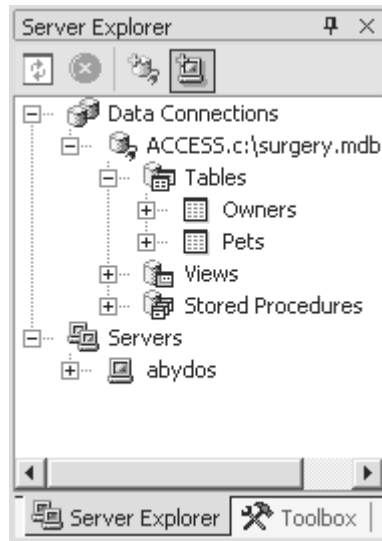


Figure 1-5. Browsing a database via the Server Explorer

4. Expand the Tables list under your database.
5. Drag and drop the table you want to use in this example onto your Service1.asmx.vb page (currently in design mode).

You'll find OleDbConnection1 and OleDbDataAdapter1 added to your service—or perhaps SqlConnection1 and SqlDataAdapter1 if you're using SQL Server. As you may remember from Tutorial 2, "Doing Databases" (*please say you've read it*), the Connection object here actually "calls" your database, and the DataAdapter talks through that connection to get information out or to modify existing data.

6. Rename your Connection object to "connMyDatabase".
7. Rename your DataAdapter object to "daMyDataAdapter".

Next, let's sort out our DataSet. Eh? A *DataSet* is basically a widget that holds the data coming back from your database. Actually, we're going to create a *Typed DataSet*, which is just a regular DataSet that runs off a template telling it about the tables and fields it will hold, thereby making our later programming much easier.

8. Right-click on your DataAdapter object and select Generate DataSet.
9. Choose the New option and enter a name of "MyData".
10. Ensure your table is checked in the list of tables to add to this DataSet.
11. Check the "Add This DataSet To The Designer" box.
12. Click on OK.

Two things just happened here. First off, MyData.xsd has been added to your project; it lists the fields that your DataSet will hold. Secondly, MyData1 has been added to your Web service. This is your typed DataSet, based on that MyData.xsd "template." (See Figure 1-6.)



Figure 1-6. Generating our typed DataSet

13. Rename MyData1 to “dsMyDataSet”.

Great! We’ve sorted the data access objects out. Next, we need to go add code to our Web service to retrieve information from this database and pass it back.

14. Press F7 to switch to the code mode for your Web service.
15. Add the following code to your Service1 class:

```
<WebMethod()> Public Function GetData() As MyData
    ' Passes populated Typed DataSet back to client
    daMyDataAdapter.Fill(dsMyDataSet)
    Return dsMyDataSet
End Function
```

Here we have a function called GetData that returns a DataSet of the MyData type, a DataSet that adheres to our MyData “template” class. The code simply fills our DataSet with information from our database table and returns it. Simple.

Oh, and of course we have that magical keyword <WebMethod()> there, too. Leave it out and this service can’t strut its stuff.

However, most databases aren’t read-only: you’ll typically want to update them, too. Next, we’re going to accept the entire DataSet back and update the backend database as required. However, in the real world, you might just want to accept individual Row objects and go from there.

16. Add the following code to your Service1 class:

```
<WebMethod()> Public Sub UpdateData(ByVal Data As MyData)
    ' Updates the backend database where needed
    daMyDataAdapter.Update(Data)
End Sub
```

Here, we’re just accepting a MyData typed DataSet and using the DataAdapter object to update our backend database. No problem. Well, that’s our whole Web service pretty much finished now and our cue to run a quick build.

17. Press Ctrl+Shift+B to build this solution.

Retrieving Smart Data

But what good is data without someone to use it? As Confucius once said, “It ain’t.” Which is why we’re about to create a neat little Windows application to discover and consume the features of our new service.

1. Create a new Windows application.
2. From the menu, select Project > Add Web Reference.
3. Type in your Web service address:
http://localhost/MyDataService/Service1.asmx.
4. Click on “Add Reference” when the page has loaded.

Next, let’s add code to test our Web service.

5. Add a button to Form1.
6. Behind the click event of your button, add the following code:

```
' Declare Service and DataSet 'template'
Dim MyWebService As New localhost.Service1()
Dim MyDataSet As localhost.MyData

' Grab data
MyDataSet = MyWebService.GetData

' Retrieve sample data -
' your fields may be different
Dim strText As String
strText = MyDataSet.Owners(0).Address
MessageBox.Show(strText)

' Update DataSet -
' again, your fields may be different
MyDataSet.Owners(0).Name = "Mr Bibbles"

' Send back to Web service
MyWebService.UpdateData(MyDataSet)
```

Okay, it might be a little more in-depth than our Reverse sample, but it's still pretty understandable. We're creating a new instance of our own local `Service1` class, then using its functionality to work with our Web service—grabbing our data, displaying one particular field, changing a little data, then running an update. Top stuff.

The special thing to note here is that, when we compiled our Web service, it recognized we were exposing a typed `DataSet` and included details of our `MyData` schema alongside, allowing us to use its objects in code as we have done here. How very clever.

7. Press F5 and test your application.

Does it all work as expected? How could we improve the interface? Well done on completing this sample!

So, what have we created in under two dozen lines of code? On the server side: a Web service that exposes data direct from your database using a typed `DataSet`. On the client side: an application that can use all the power of your “disconnected” data in just a few lines of code—without having to worry about the data access code in the slightest.

This is smart data. It's a lot more than just a single `True` or `False` return value, and its simplicity can really help speed up your application development. And what do you know—it doesn't stop with `DataSets`. You can expose data-holding classes, structures, and arrays through your Web services, too. Just use them and let Visual Studio .NET do the rest.

Tsk, can you spell *clever chuff*?

The Top Five Distribution Tips

So, you've written the greatest Web service since `SlicedBread.asmx`, and now you want to roll it out to the world? Yes, you could just take what you've done so far and follow the deployment instructions in the next section, but five quick changes will give your service that professional touch, mini alterations that should precede the distribution of *any* Web service.

Get out that notebook. It's time to talk turkey.

1. *Rename your class.*

Nobody likes dealing with a Web service called `Service1`, so, before you redistribute, make sure that you rename your class from something like `Public Class Service1` to something like `Public Class Customer`. Or something. Ahem.

2. *Christen your ASMX file.*

You've renamed your Web service so it sounds all professional, but your clients are still accessing your Service via the `http://localhost/Customer/Service1.asmx` file. Eugh. To combat this, simply rename your ASMX file by right-clicking on it in the Solution Explorer and selecting Rename.

3. *Add a service description.*

What does your Web service do? Provide access to your customer database or allow you to send SMS messages? Well, you could just let your clients guess. However, a more user-friendly technique would be to add a description to the `<WebService(>` attribute. So, before distributing, change your class as follows:

```
<WebService(Description:="This is a class that does amazing things with data.")> _
Public Class AmazingData
    ' ... etc ...
End Class
```

This description is used in the automatically generated Web interface and is automatically absorbed by all clients using your service and utilized in the Visual Studio .NET programming environment.

4. *Add method descriptions.*

Just as your service can include a description, so can your individual methods and functions. Again, this is used by the Web interface and Visual Studio .NET. And, again, it's simple to implement: just add a description to each `<WebMethod(>` attribute, like so:

```
<WebMethod(Description:="Returns the current server date and time.")> _
Public Function GetServerDate() As Date
    Return Now
End Function
```

5. Change the namespace.

You've already seen the .NET Framework organizing functionality into "namespaces." They're just unique identifiers for a certain set of functionality. That's all a namespace is here, too: a unique string that identifies this Web service. By default, it's `http://www.tempuri.org/`, and you will need to change this to a unique value. The namespace you provide doesn't necessarily have to point to anything on the Web, but it's recommended that you at least own the domain you use (I mean, *purrr-lease*). So, change the namespace to something unique before unveiling your service by altering the `Namespace` property of the `<WebService(>` attribute, as so:

```
<WebService(Description="Yadda", _
Namespace="http://www.amazingdata.com/query/")> _
  Public Class AmazingData
    ' ... etc ...
  End Class
```

Deploying Your Web Service

Once you've finished your Web service, you can stick it literally anywhere. And, resisting another crude Bill Gates prod, I'm talking about the Internet or your network.

How? Well, if you built your Web service on the server it will be running on, you don't actually need to do anything else. When you build your service in Visual Studio .NET, it's automatically deployed for you.

However, if you want to move it over to another machine, you'll need to use the Copy Project feature. Simply open your Web service in Visual Studio .NET and select `Project > Copy Project` from the menu. Change the Project Destination folder to the new HTTP home of your service and ensure that the "Copy: Only files needed to run this application" option is selected. Then click on the OK button.

TOP TIP *You've seen that Web services run over HTTP. In fact, they work in just the same way as regular ASP.NET Web applications, which means that you can administer them via the Internet Services Manager plug-in: to launch it, select `Start > Program Files > Administrative Tools > Internet Services Manager`. Useful for deleting all those Hello World samples!*

Alternatively, if you're deploying via FTP to a .NET host, copy across your ASMX pages, Web.config and VSDISCO files. You'll also want to transfer your compiled DLL assembly from your local Bin subdirectory to a Bin subdirectory on the server. Basically, copy across anything that isn't a Visual Studio .NET project file, in the exact same way as you would with an ASP.NET Web application.

You can also upload direct to a supported .NET host by clicking the Web Hosting link on the Visual Studio .NET Start page.

And that's a rap, folks: your Web service is now exposed to the world. Let the consuming commence!

FAQ

Still got a couple of questions about Web services? Looking to learn from the mistakes of others? Let's review our list of frequently asked questions . . .

In your examples, you've worked on a Web service project with just one Service class. Can you have multiple Service classes in one Web service project?

Absolutely! Simply add another Web service to the project, by selecting Project > Add Web Service from the menu. You can even add a Web service to a regular ASP.NET Web application in exactly the same way.

My code always seems to have something wrong with it. Is there any way to debug a Web service, just as you would a Web form or a regular Windows application?

Yes—literally *just* as you would a Web form or regular Windows application! Simply move to the line you wish to break on and press the F9 key to turn debugging on. Next, run your service live by pressing F5—then either run the application that uses this Web service or access its operations via the Web interface. When the line you highlighted is about to run, your code will break and you should be able to step through line by line using F8. Happy bug hunting!

What happened to all those ASP.NET objects, like Application and Session?

Well, because Web services are a part of ASP.NET, objects such as Application, Session, Server, User, and all their merry friends are still at your disposal (though admittedly not used here quite as often). Check out the Help index or Tutorial 3, "Working the Web" to find out more.

I've added a few new methods to my Web service. Have I now got to update all my clients?

Don't worry—as long as you haven't "broken" any of your existing methods, you won't have any problems. If, however, you want to use these new methods in your applications, open the client project and, using the Solution Explorer, right-click on the Web reference and select Update Web Reference. Your new methods will be discovered, and you'll be instantly able to use the new features in code.

I'm wanting to expose some pretty sensitive information via my Web service. Do these things do security?

You bet your bottom dollar, kiddo. You can either implement your own authentication by adding username and password arguments to your methods, or use one of the ASP.NET authentication options, presenting your credentials before you begin using the service. For more information, look up "Web services, security" in the Help index for the full lowdown.

I like to know my file extensions. What does ASMX stand for?

Despite many leading publications revealing that "it doesn't have any real meaning," I can tell you that ASMX stands for *active server methods*, with the X just referring to the next generation of development techniques (that is, old ASP pages are now ASPX pages).

Good news! My Wonderful Weather Web service has grown to great heights. Bad news! My code runs a powerful algorithm to correctly predict whether it rains or shines, and that takes time. Now that I'm getting requests every second or so, my servers are clogging up. What can I do?

It's a little-known fact, but you can actually "cache" what a Web service gives out and automatically serve up the same response next time. How? Simply by specifying a CacheDuration in the WebMethod attribute, like this:

```
<WebMethod(CacheDuration:=60)> _
Public Function GetWeather() As String
    ' ... complicated code ...
End Function
```

Here, the Web server will cache and serve up the same results for GetWeather sixty seconds after the last query. You can test this baby by returning the time. See what happens?

Our company has just created a whole bundle of Web services containing company-wide functionality. The project is proving exceptionally popular among the developer team—and, to cope with demand, we're going to have to transfer the service to more powerful servers. Won't this break the programs currently using our Web service?

If your existing clients are looking to access a Web service and it isn't there, yup, their applications will either raise an error or die a dramatic death. Either way, it ain't good.

However, it's relatively simple to change the server name of your Web service. In Visual Studio .NET, simply open your project and select your Web service under the Solution Explorer, Web References folder. You should notice a few items in the Properties window—including Web Reference URL. Change this property to point to your new Service, then recompile your application.

If your Web service often changes location, it might be worthwhile changing the URL Behavior property to Dynamic. If you're working on a Web application, this adds a line containing the URL to Web.config (or App.config for Windows applications).

If changes occur in the future, you'll no longer have to redistribute all your files again—just send out an updated .config file. Top stuff!

WHERE TO GO FROM HERE

Looking to learn more about Web services? You're not alone—and, thankfully, the world is starting to brim with resources for the new technology. Helping you wade through the rubble, here's my pick of the best:

- *Architecting Web Services* (Apress, ISBN 1-893115-58-5): If you're looking for a full-on book delivering nothing but Web service information, this is the title for you. A real work of art.
- Other books: *Professional ASP.NET Web Services* (Wrox, ISBN 1-861005-45-8).
- Visual Studio .NET Help: The Visual Studio .NET documentation provides a full walkthrough of Web services in a data-driven environment. Definitely worth a browse. On the Programs menu, select Microsoft Visual Studio .NET > Microsoft Visual Studio .NET Documentation.
- www.xmethods.com: Showcasing examples of Web services in action, this site is essentially the mini *Yahoo!* of the ASMX world. Current listings include an SMS service, a dictionary, and a horoscope service.

- www.vbws.com/newsletter/: Want the latest news on Web services? Trainer Yasser Shohoud publishes his own newsletter giving the full lowdown, each and every month. Subscribe for free here.
- www.dotnetjunkies.com: Incredibly useful .NET developer site, with pages dedicated to creating Web services. Tips, tricks, code samples, and more at this site. Definitely worth a surf.
- www.asp.net: Only Microsoft could own such an exclusive address. Lists all the latest books, community sites, and code samples.
- www.vbxml.com: Your one-stop shop for learning more about XML, the markup language behind Web services. Also includes numerous .NET code snippets.
- www.webservices.org: Providing industry news and details on Web services implemented on *all* platforms. On the whole, a pretty drab site—but, if you're into the likes of Linux and Java, a decent resource.
- www.dnj.com: Homepage of the popular Microsoft-endorsed programming magazine, *Developer Network Journal*. Regularly features articles on implementing Web service technologies. Plenty of online content too, plus the option to subscribe (plug, plug).
- microsoft.public.dotnet.framework.aspnet.webservices: The official Microsoft newsgroup for Web services discussion. For speedier answers, try searching the archives first at www.googlegroups.com.

CONCLUSION

When asked exactly what .NET is, Microsoft's official answer is always “.NET is the Microsoft Web services platform.”

In other words, Web services are at the heart of Microsoft's vision for a distributed computing future. Yes, there's a lot of hype—but there's no denying that Web services are cooler than a Pepsi-drinking polar bear. And, over the last twenty or so pages, you've learned how to get your own up and running in minutes.

We launched the tutorial today by creating our own mini Web service, then looked at using its operations from a Web page—plus found out how to “discover” and “consume” that functionality from within our applications. After that, we talked about its possibilities, plus went on to access exceptionally “smart data” in under twenty lines of code.

Finally, we checked out deployment and the top five things you really should do before making your service available to the world. And finally-finally, we reviewed all those frequently asked questions alongside a list of resources to take your knowledge to the next level.

Just remember: the next time you need to get your computers more talkative than a Jerry Springer audience, think Web services. They get computers exchanging data, they distribute your application, they solve problems, they're just cool.

But that's all from me—until the next time, this is Lrak Eroom signing off for tonight, wishing you a very pleasant evening, wherever you are in the world. Goodnight!

CHAPTER 6.1

INTRODUCING THE WORLD OF WEB SERVICES REVIEW SHEET

- Web services allow you to get computers talking to each other. The process is often compared to the old DCOM; however, it is loosely coupled and based on open standards.
- On the server side of a Web service, you have chunks of compiled code that are exposed to the client. On the client side, you have an application or browser making HTTP requests to those code chunks. The .NET Framework passes data between requests about in XML (eXtensible Markup Language).
- To generate your own Web service, create a new ASP.NET Web service project in Visual Studio .NET. By default, you will have a Web service called Service1.asmx added to your application. You can add new Web services to a project by selecting Project > Add Web Service from the menu.
- To author the code behind your Web service, right-click on the service in the Solution Explorer and select View Code.
- By default, none of your methods, functions, or properties are exposed as part of your Web service. To expose a chunk of code, prefix it with the <WebMethod(> attribute, as so:

```
<WebMethod(> Public Function GetDate() As Date
    Return Now
End Function
```

- You can view your Web service in a browser by simply building the project, then visiting its dynamically generated ASMX page. Another technique is to simply press the F5 key while working on your Web service project.
- To “discover” a Web service for use in your Visual Studio .NET application, select Project > Add Web Reference from the menu. Specify the address of your Web service, press the Enter key, then select Add Reference. Alternatively, browse the Microsoft UDDI Directory at <http://uddi.microsoft.com/visualstudio>.

- To “consume” the functionality of a Web service in your application, simply treat it as you would any regular object. The XML is all parsed out and handled for you. For example, the following demonstrates a discovered Web service in use:

```
Dim MyObject As New LocalHost.Service1()  
MessageBox.Show(MyObject.GetDate)
```

- Your application doesn’t care where it accesses Web services from. The data could be coming from the local computer, across a network, or from the other side of the world via the Internet. So long as that connection is available, the Web service works.
 - To distribute your Web service to another server, from the menu select Project > Copy Project. Alternatively, copy the ASMX, CONFIG, and VSDISCO files across to the new Web Application directory, plus create a subdirectory called Bin, copying your compiled Bin/ProjectName.DLL assembly into it.
 - Web service members can accept and return more than just base data types. They can take in and pass back anything, including DataSets, data-holding classes, and user-defined types (structures).
-

Index

Symbols

& concatenation character, 529, 549
#Region and #End Region keywords, 13, 454
(Base Class Events), 29, 49
.ExecuteScalar function, 214, 218
.NET
 background and components, 447–450
 hosting accounts, 310
 objects and COM, 513–514
 managed and unmanaged code, 110
.NET Framework
 client browser and, 275
 components (controls), 9
 defined, 4–5, 21, 449–450
.Show method (forms), 93
_VIEWSTATE field, 253, 275, 287–288, 359, 361–362
< symbol, 46
<> symbol, 46
<WebMethod(>) attribute, 423, 429, 443
<WebService> attribute, 436
> symbol, 46

A

abstraction (OOP), 409, 462
Access connection with VB .NET, 126–133
 DataSet control, 130–131
 displaying and updating data, 131–132
 selecting providers, 126
 talking to an Access database, 129
 testing the Access connection, 127
 viewing Access tables, 128
Access database, creating. *See* Microsoft Access database, creating
Access-generated SQL statements, 170–171
Access generic code template, 187–188
Access Jet database, 122, 135
ACID transaction theory, 211
ASCII key value, 483
Active Server Pages (ASP). *See* ASP.NET Web applications
ActiveForm property, 341, 491
ActiveForm property (mobile), 324
ActiveLinkColor property, 31
ActiveX, 451–452, 512
Add function, 86–87
Add module, 94
Add Web References, 425–426, 429
Add Web Service, 438
Add Windows Form, 93
AddressOf keyword, 466
ADO.NET
 DataSet, 513
 defined, 172
 reference info, 236–237
AdRotator Image control, 269, 305–306
advertisement images (mobile), 338
AdvertisementFile property, 269
AdvertisementFile property (mobile), 338
advertisements, adding to sites, 305–306
algorithms, cryptography, 516
AlternateFormat property (mobile phone), 334
AlternateText property (mobile), 334
Anchor property, defined, 40
apostrophes (SQL statements), 210, 235
AppendText function, 32
Application directory, finding, 488
Application object, 278, 313, 529
applications
 accessing Web services from, 425–429
 compiling and distributing, 471–472
 ending, 486
 upgrading existing, 472
Applied Device Filters, 350, 353
Argument property, 353
ArrayList object, 528
arrays
 alterations, 458
 creating and sorting, 523–525
 using special functions, 526–527
As New declaration, 80, 93
As New statement, 80
ASMX (active server methods), 439
ASMX files, 436
ASP.NET. *See also* Web services applications, fine-tuning, 529
 caching, 313

- ASP.NET (*continued*)
 - Copy Project feature, 310
 - error messages, 311
 - files to FTP, 318
 - objects, 438
 - Web service, defined, 7
 - ASP.NET Web applications, 243–276
 - background and basics, 243–244
 - controls, 258–272
 - AdRotator Image, 269
 - Calendar, 271
 - CheckBox, 267–269
 - CheckBoxList, 267–269
 - CompareValidator, 262–264
 - CustomValidator, 263
 - DataGrid, 265–266
 - DataList, 265–266
 - DropDownList, 260–261
 - Hyperlink, 259–260
 - Image, 269–270
 - ImageButton, 270–271
 - LinkButton, 259–260
 - ListBox, 260–261
 - RadioButton, 268–269
 - RadioButtonList, 268–269
 - RangeValidator, 263
 - RegularExpressionValidator, 263
 - Repeater, 265–266
 - RequiredFieldValidator, 262–264
 - ValidationSummary, 264
 - XML, 272
 - defined, 7
 - Solution Explorer, 246
 - viewing data on Web, 154–155
 - ASPX
 - pages, 246, 324
 - Web forms, 244
 - assembly files, 246
 - Assembly Registration Tool, 513
 - attributes, 457, 468–469
 - authentication, 294–303
 - coding for user authentication, 298–301
 - compiling/testing finished application, 301–302
 - Forms authentication, 294, 317
 - Passport authentication, 294, 317
 - RedirectFromLoginPage method, 317
 - setting up Forms authentication, 295–298
 - expanding authentication, 303
 - types of, 294–295
 - Web services, 439
 - Web.config file and security, 311
 - Windows authentication, 294, 317
 - Auto Format link, 157
-
- B**
 - BackgroundImage properties, 34
 - (Base Class Events), 29, 49
 - base classes, 57, 460–461
 - BeginUpdate function, 36
 - Bin folder, 17, 21, 310
 - binding data. *See* data binding
 - BitArray object, 528
 - block-level scoping, 459
 - BMP (bitmap) images, 354, 365
 - books, reference. *See* reference books
 - Boolean data types
 - default setting, 85
 - defined, 55, 69
 - Boolean values, defined, 34
 - Bootstrapper option, 107, 110–111
 - brackets, use of, 14
 - BringToFront function, 30, 35
 - Browser device property, 347
 - browsers
 - logging code, 281–282
 - OpenWave Mobile Browser, 362
 - Build menu, 17, 21
 - business logic layers, 403
 - Button control, 9–11, 31
 - ButtonClick event, 491
 - ByRef (by reference), 85, 409–410, 459
 - ByVal (by value)
 - default, 459
 - keywords, 85
 - passing data, 409
-
- C**
 - CacheDuration, 439
 - caching ASP .NET applications, 313, 529
 - Calendar control, 271
 - Calendar properties, 254–255
 - CalendarEntryText property (mobile), 338
 - CanInitiateVoiceCall device property, 347, 349
 - CanSendMail device property, 347
 - cascade updates and deletes, 143–144
 - cascading style sheets (CSS), 307–309
 - case sensitivity, 45
 - Catch blocks, 98–99, 115
 - CauseValidation property, 264
 - characters. *See also* symbols
 - & concatenation character, 529
 - capitalizing, 486
 - colon, 482–483
 - special code, 482–483
 - underscore, 60, 482–483
 - CheckBox control, 32–33, 267–269, 503
 - CheckBoxList control, 267–269
 - Checked property, 33

- CheckedChanged event, 33
- CheckedListBox control, 503
- CheckState property, 33
- child forms (MDI), 491
- cHTML (Compact HTML), 341, 354, 361
- Class keyword, 462
- class libraries
 - vs. components, 410–411
 - defined, 462
 - DLL, 403, 404–405
 - project defaults for, 537–538
- Class Name box, 27–29
- classes. *See also* objects and classes
 - adding enumerations to, 378–381
 - adding events to, 383–385
 - adding methods, functions, parameters to, 382–383
 - adding properties to, 376–378
 - building class libraries, 391–397
 - class library test project, 397–401
 - collection, 529
 - as data holders, 402
 - as forms in VB .NET, 453–454
 - multiple instances of, 374
 - naming conventions, 372
 - PageSettings class, 521
 - prefixes for, 372
 - Private/Public content in, 377, 390
 - renaming, 435
- classes (listed)
 - File class, 498–499
 - FormsAuthentication class, 298
 - Garbage Collection class, 408
 - Hashtable class, 529
 - PageSettings class, 521
 - PrintDocument class, 521
 - PrinterSettings class, 521
 - PrintPreviewControl class, 521
 - Random class, 65–66
 - StringBuilder class, 529, 549
 - System.Random class, 481
- Click event, 30, 35
- Clipboard object, 498
- clipboard, storing to, 498
- Close menu item, 81–83
- code
 - accessing databases in, 171–186
 - adding core data access code, 175–181
 - adding/deleting rows, 181
 - adding relationships to DataSets, 181–186
 - declaring objects, 174–175
 - designing application interface, 172–173
 - using Command objects, 172
 - using DataSets, 172
 - coding errors in VB .NET, 455
 - collapsible, 454
 - exposing, 443
 - managed and unmanaged, 110
 - regions, collapsing and expanding, 13
 - storing often-used, 514
 - templates for DataSet relationship, 188
- code examples. *See also* coding techniques
 - Access generic code template, 187–188
 - adding/clearing items on
 - DropDownList control, 260
 - adding parameters to constructors, 464
 - arrays
 - creating and sorting, 523–525
 - declaring, 458
 - special functions, 526–527
 - authenticating visitors, 317
 - binding data in Web forms, 156
 - binding to DataGrid, 135
 - Button control, 31
 - ByRef (by reference), 409–410
 - CacheDuration, specifying, 439
 - calling method with parameters, 94
 - capitalizing characters, 486
 - Catch blocks, 115
 - CheckBox control, 33
 - checking Selected property, 267–268
 - checking User object, 317
 - Collection classes, using, 415
 - Collection object functionality, 527–528
 - colon and underscore characters, 483
 - COM objects, using, 512
 - ComboBox control, 37
 - command line parameters, reading, 509–510
 - cookies
 - removing encrypted, 317
 - setting and reading, 292
 - custom mobile device filter, 353
 - customizing cursors, 482
 - DataSet relationship, template for, 188
 - declaring events in class, 390
 - default properties, 463
 - delegates, 466–467
 - Delete statements (SQL), 187
 - deleting records from DataSet, 154
 - discovered Web service in use, 444
 - displaying form, 93
 - Dispose and Finalize methods, 408

- code examples (*continued*)
 - Do-Until loop, 70
 - encrypting text strings, 515
 - encryption functions, 516–519
 - ending applications, 486
 - enumeration and public variable, 390
 - ErrorProvider, use of, 500
 - exposing code (Web services), 443
 - File class, uses of, 498–499
 - filter definition, 365
 - find IndexNumber of current record, 154
 - finding application directories, 488
 - finding last day of month, 497
 - folder locations, getting, 510
 - For-Next loop, 69
 - formatting dates, 488–489
 - function sample, 93
 - functions, expanding program with, 84
 - generating random numbers, 481–482
 - Get and Set blocks, 389
 - GroupBox control, 34
 - GUIDs, creating, 508
 - Handles keyword, 407, 497
 - If-Then statements, 108
 - IIF function, 500
 - inheritance, 469
 - Insert statements (SQL), 187
 - IsValid function, 501–503
 - IsValid property, 264
 - With keyword, 479
 - keywords as variable names, 506
 - Label control, 30
 - LinkLabel control, 31
 - ListBox control, 36
 - listing/setting fonts, 485
 - MDI (Multiple Document Interface) applications, 491
 - method (subroutine), 93
 - method that accepts parameters, 94
 - mobile Web forms, moving between, 341
 - MonthCalendar control, 37
 - navigate/manipulate records, 164
 - number-only text box, 483
 - object declaration using WithEvents, 390
 - OLE DB strings, generating, 508
 - one-user session data storage, 365
 - Optional keyword default, 407
 - overloading, 465–466
 - parameters and SQL statements, 235
 - passwords and user names, 233
 - passwords, generating memorable, 519–520
 - PictureBox control, 35
 - previous instances, checking for, 510–511
 - printing from applications, 521–523
 - property declaration, 458
 - public variable declaration, 94
 - query string for storing user data, 359–360
 - RadioButton control, 33
 - read-only property declaration, 463
 - ReadFromRegistry function, 511
 - ReadTextFromFile function, 495–496
 - ReadXML /WriteXML methods, 494
 - Recordset into DataSet, 513
 - replacing text, 484–485
 - reports in applications, 239
 - resetting forms, 504
 - Resume Next statement, 115
 - retrieving data into DataSet, 135
 - retrieving numbers with Val, 486
 - reversing strings, 484
 - rounding numbers, 484
 - Select statements (SQL), 187
 - SelectedIndex property, 261
 - setting/retrieving data from clipboard, 498
 - setting variable value, 69
 - shared members, 467–468
 - static variables, 507
 - strings
 - declaring, 457
 - trimming, 485
 - text case conversion, 498
 - TextBox control, 32
 - three-character prefixes, 69
 - TODO keyword, 480
 - Try-Catch-Finally blocks, 115
 - update database changes, 135
 - Update statements (SQL), 187
 - whole numbers, checking for, 509
 - WriteTextToFile method, 496
 - WriteToRegistry function, 511–512
 - XML document sample, 492–493
 - XML files in applications, reading, 493–494
- coding techniques, 278–292
 - browsers, logging user, 281–282
 - cookies, setting, 279–280
 - files, uploading, 284–285
 - page Postbacks, determining, 283
 - query strings, using, 285–287
 - user Referrers and URLs, finding, 282
 - user sessions, remembering, 280
 - users IP addresses, grabbing, 283

- users, redirecting, 278–279
 - ViewState, adding to, 287–288
- Collection classes, 393–396, 415, 529
- Collection objects, 395
- collections in .NET, 527–528
- colon character, 482–483
- columns and fields, 233
- COM Callable Wrapper (CCW), 410
- COM Interop service, 512
- COM objects, 110, 512
- ComboBox control, 36, 176, 503
- Command and DataReader objects, 206
- Command control (mobile Web applications), 325
- command line parameters, 509–510
- Command mode, 490
- Command objects, 172, 217–218
- Command window, 490
- Common Language Runtime (CLR), 18, 110, 471, 548–549
- CompareValidator control, 262–264
- compiling programs, 17–18
- components, 471
- computer loops, 65–67, 69
- connection objects, 203–204
- connection strings, 174
- constructor code, 386, 390
- constructors, 464
- controls, 23–38, 258–272 *See also* ASP.NET Web Applications, controls
 - accessing events supported by, 28
 - ActiveX controls, 512
 - AdRotator Image control, 269, 305–306
 - Button control, 31
 - Calendar control, 271
 - CheckBox control, 32–33, 267–269, 503
 - CheckBoxList control, 267–269
 - CheckedListBox control, 503
 - ComboBox control, 36, 503
 - CompareValidator control, 262–264
 - CustomValidator control, 263
 - DataGrid control, 265–266
 - DataList control, 265–266
 - DateTimePicker control, 503
 - DomainUpDown control, 503
 - DropDownList control, 260–261, 312
 - dynamic, 263
 - ErrorProvider control, 499–500
 - Form control, 287
 - GroupBox control, 34
 - HTML, 247
 - Hyperlink control, 259–260, 278, 312
 - Image control, 269–270
 - ImageButton control, 270–271
 - ImageList control, 491
 - introduction to, 23–24
 - invisible, 49
 - Label control, 30
 - LinkButton control, 259–260
 - LinkLabel control, 30–31
 - list type, 260–261
 - ListBox control, 35–36, 260–261, 312, 503
 - methods and functions, 26–27
 - Mobile Web Forms, 330–339
 - AdRotator control, 338
 - Calendar control, 338
 - Command control, 333
 - DeviceSpecific control, 337
 - Form control, 330
 - Form control (mobile), 324
 - Image control, 334
 - Label control, 331
 - Link control, 333
 - Link control (mobile), 324
 - List control, 335
 - Mobile QuickStart tutorial, 336
 - ObjectList control, 336
 - Panel control, 331
 - PhoneCall control, 334
 - SelectionList control, 335–336
 - StyleSheet control, 337
 - TextBox control, 332
 - TextView control, 332
 - Validation controls, 339
 - MonthCalendar control, 37–38, 503
 - Name properties, 24–25
 - NumericUpDown control, 503
 - PictureBox control, 35
 - prefixes for, 542–544
 - RadioButton control, 33–34, 268–269, 503
 - RadioButtonList control, 268–269
 - RangeValidator control, 263
 - reading/writing properties in code, 24–27
 - RegularExpressionValidator control, 263, 500–501
 - Repeater control, 265–266
 - RequiredFieldValidator control, 262–264
 - resizing at runtime, 478–479
 - SaveFileDialog, 63
 - server-side controls, 529
 - static, 263
 - TextBox control, 32, 503
 - ToolBar control, 491
 - validation, 261–264
 - ValidationSummary control, 264

- controls (*continued*)
 - Web Forms, 247
 - Windows Forms defaults (listed), 545–547
 - XML control, 272
 - ControlToValidate property, 263
 - cookies
 - code for setting, 279–280
 - encrypted, 317
 - mobile devices and, 359, 365–366
 - setting and reading, 292
 - Cookies device property, 347
 - Copy Project feature, 310, 437
 - Crypt (encryption), 514–515
 - cryptography algorithms, 516
 - Crystal Decisions application, 222
 - Crystal Reports. *See also* reports, designing, 220–224
 - CrystalReportViewer, 221, 239
 - CSS (Cascading Style Sheets) tutorial, 309
 - Cursor property, 31, 482
 - cursor, customizing, 482
 - Custom Actions and Launch Conditions menus, 108
- D**
- data access layers, 403
 - data binding
 - summarized, 163–164
 - on the Web, 154–159
 - adding typed DataSet, 156
 - creating ASP.NET Web application, 154–155
 - displaying data, 156–157
 - filling and binding DataSet, 157–158
 - in Web forms with DataGrid, 156
 - to Windows forms, 145–154
 - binding text boxes to data, 150–152
 - deleting records, 154
 - moving around rows, 152
 - typed and untyped DataSets, 147–149
 - updating database, 152–153
 - Data Form Wizard, 159–160, 164
 - data types
 - Boolean, 55
 - Short, 55
 - SQL Server (listed), 550–551
 - String, 55
 - VB .NET (listed), 548–549
 - data warehouses, 232
 - DataAdapter objects, 175, 182
 - Database Diagrams, 197–199
 - databases
 - accessing in code, 171–186
 - adding and deleting rows, 181
 - adding core data access code, 175–181
 - adding relationships to DataSets, 181–186
 - declaring the objects, 174–175
 - designing application interface, 172–173
 - using Command objects, 172
 - using DataSets, 172
 - binding on the Web, 154–159
 - adding typed DataSet, 156
 - creating ASP.NET Web application, 154–155
 - displaying data, 156–157
 - filling and binding DataSet, 157–158
 - connecting to Access with VB .NET, 126–133
 - DataSet control, 130–131
 - displaying and updating data, 131–132
 - selecting providers, 126
 - talking to Access database, 129
 - testing Access connection, 127
 - viewing Access tables, 128
 - data binding to Windows forms, 145–154
 - binding text boxes to data, 150–152
 - deleting records, 154
 - moving around rows, 152
 - typed and untyped DataSets, 147–149
 - updating database, 152–153
 - Data Form Wizard, 159–160, 164
 - definition and basics, 120–121, 135
 - Microsoft Access database, creating, 130–145
 - adding name table, 141
 - adding relationships, 141–145
 - field names, 138
 - primary keys, 139
 - SQL, asking questions in, 165–171
 - Access-generated SQL statements, 170–171
 - Datasheet view, 167–168
 - SQL statements, 168–169
 - SQL view, 167
 - SQL Server, creating, 191–200
 - adding relationships, 197–200
 - designing database tables, 194–197

- linking tables, 198–199
 - Server Explorer, 192
 - Web application, creating, 200–215
 - adding data, 206–210
 - adding transactions, 210–215
 - Command and DataReader objects, 206
 - designing Web page, 201–202
 - setting up connection objects, 203–204
 - writing the code, 202–206
 - Web sites (database references), 236–237
 - DataBindings property, 164
 - DataGrid, binding to, 135
 - DataGrid control, 131–132, 156, 164, 265–266
 - DataList control, 265–266
 - DataReader, 233
 - DataReader object, 205, 218
 - DataSet control, 130–131
 - DataSets, 172, 175–176
 - adding relationships to, 181–186
 - filling and binding, 157–158
 - multitable, 430
 - one for all pages, 313–314
 - for reports, 225
 - retrieving data into, 135
 - storing, 529
 - typed, 147–149, 430
 - untyped, 147
 - XML and, 234, 494
 - DataSource property, 265
 - Date data type, defined, 55, 69
 - dates
 - DateTime (data type), 195
 - finding last day of month, 496–497
 - formatting, 488–489
 - DateTimePicker control, 503
 - DCOM, 452
 - debugging, 44
 - default, 311
 - Web services, 438
 - Decimal data type, 195, 457
 - decision making. *See* If statements
 - decrypting files, 516
 - Default keyword, 463
 - default properties, 463
 - default.aspx, 201, 480–481
 - defaults
 - Visual Studio .NET projects (listed), 536–541
 - Windows Forms controls (listed), 545–547
 - delegates, 466–467
 - Delete statements (SQL), 168, 187
 - deleting records, 154
 - deployment
 - of Web applications, 309–310
 - of Web services, 437–438
 - DES (Data Encryption Standard), 516
 - design time, defined, 11
 - Details, Report, 224, 239
 - Device object, 344, 365
 - deviceFilters element, 349
 - DialogResult variable, 61
 - Dim (declaring in memory), defined, 54, 69
 - Dim statement, 89
 - DISCO files, 427
 - Display property, 263
 - Dispose method, 408
 - distributed computing, 448
 - distribution of programs
 - setup programs, building, 100–108
 - adding shortcut to Programs menu, 104–105
 - adding subfolder to Programs menu, 105
 - Bootstrapper option, 107
 - customizing setup, 107–108
 - Primary Output file, adding, 103–104
 - setting Properties, 107
 - testing setup program, 105–106
 - XCOPY deployment, 100
 - distribution (Web services), 435–437, 444
 - DNS names, 283
 - Do-Until loop, 70
 - Dock property (controls), 479
 - Document property, 272
 - DocumentContent property, 272
 - DocumentSource property, 272
 - DomainUpDown control, 503
 - DrawGrid property, 478
 - drawing tools, 495
 - DropDownList control, 260–261, 312
 - DropDownStyle property, 36
 - dynamic control, 263
- ## E
- e-mail addresses, validating, 500–503
 - “e” parameter, 491
 - Else and ElseIf statements, 45
 - EnableClientScript property, 264
 - EnableViewState property, 288, 362
 - encapsulation (OOP programming languages), 370–371, 409, 462
 - encryption
 - Crypt, 514–515
 - of data, 312, 516
 - of files, 516–519

- End If statements, 108–109
- EndWith statements, 479
- Enter event, 32
- Enterprise Developer and Architect, 533
- Enterprise Services, 528
- enumerations (Enum), 378–381, 390, 406
- error messages, 109, 311
- ErrorProvider control, 499–500
- errors, handling
 - exception handling, 460
 - Exception Objects (Ex), 99, 115
 - Resume Next statement, 96–98
 - review and summary, 115
 - runtime errors, 96–100
- Event keyword, 467
- events
 - (Base Class Events), 29, 49
 - ButtonClick event, 491
 - classes and, 390
 - defined, 27
 - responding to, 27–29
- events (listed)
 - CheckedChanged, 33
 - Click, 30, 35
 - Enter, 32
 - KeyDown, 32
 - KeyPress, 32
 - KeyUp, 32
 - Leave, 32
 - LinkClicked, 31
 - MouseMove, 30, 34
 - RemoveBoldedDate, 37
 - SelectedIndexChanged, 36, 261, 312
 - SelectedValueChanged, 36
 - SetDate, 37
 - SetSelectionRange, 37
 - TextChanged, 32, 36
- exception handling (errors), 460
- Exception Message property, 99
- Exception object (Ex), 99, 115
- .ExecuteReader function, 205
- .ExecuteScalar function, 214, 218
- exercises
 - Access, connecting to with VB .NET, 126–133
 - DataSet control, 130–131
 - displaying and updating data, 131–132
 - selecting providers, 126
 - talking to Access database, 129
 - testing Access connection, 127
 - viewing Access tables, 128
 - advertisements to sites, adding, 305–306
 - class libraries, building, 391–397
 - class library test project, creating, 397–401
 - classes and objects, creating, 372–376
 - coding for user authentication, 298–301
 - compiling programs (basics), 17–18
 - complex Web application, creating, 248–258
 - Crystal Report, creating reports with, 220–224
 - data to reports, adding, 227–229
 - databases in code, accessing, 171–186
 - enumerations to class, adding, 378–381
 - events, responding to, 27–29
 - events to class, adding, 383–385
 - filters in mobile applications, 349–352
 - first program, creating, 5–12
 - form-level variables, declaring, 54–56
 - forms authentication, setting up, 295–298
 - grouping report data, 230–232
 - image formats, determining, 354–358
 - interfaces, designing with FrontPage, 304–305
 - menus, adding, 72–75
 - message boxes, displaying, 12–16
 - MessageBox function (multiple choices), 59–61
 - method: “Goodbye” subroutine, creating, 81–83
 - methods, functions, parameters (classes), 382–383
 - Microsoft and System namespaces, exploring, 57–59
 - mobile Web applications, creating, 322–330
 - multiple forms to application, adding, 77–80
 - Northwind database, exploring, 122–125
 - office lottery application (loops/random numbers), 64–67
 - password program, creating, 56–57
 - Picture Viewer application, creating, 38–44
 - adding code, 41–42
 - testing program, 42–44
 - properties to class, adding, 376–378
 - reading/writing properties in code, 24–27
 - Resume Next statement, implementing, 96–98
 - setup programs, building, 100–108

- simple Web application, creating, 244–246
- smart data, passing (Web services), 430–433
- smart data (Web services), retrieving, 434–435
- SQL Server database, creating, 191–200
- Start function launching routine, 61–62
- styles, defining and using, 307–309
- target mobile devices, determining, 343–348
- text string variable, creating, 52–53
- Try-Catch-Finally blocks, 98–100
- Web application database, creating, 200–215
- Web service from applications, accessing, 425–429
- Web services, creating, 422–424
- Windows application, creating, 452–456
- writing to files, 62–63
- Exit method, 486
- exiting code, 482
- F**
- Field Explorer, 228
- field names, 138
- fields
 - adding to reports, 227
 - changing display format (reports), 231
 - columns and, 233
 - data-holders, 389
 - database (defined), 121
- File class, 498–499
- File Field element, 284
- File System screen, defined, 107
- File types screen, defined, 107
- files
 - manipulating, 498–499
 - reading and writing to, 495–496
 - writing to, 62–63
- FileStream object, 62
- Filter function (arrays), 526–527
- filtering strings, 41
- filters, creating custom, 353
- filters (listed)
 - Applied Device Filters, 350, 353
 - isEricssonR380 filter, 353
 - isHTML32 filter, 349
 - isNokia7110 filter, 353
 - prefersGif filter, 365
 - prefersWBMP filter, 365
 - supportsVoiceCalls filter, 349
- filters (mobile applications), 348–353
 - displaying images using, 354–358
 - how and when to use, 353
 - using filters (example), 349–352
- Finalize method, 386–387, 408, 464–465
- Finally blocks, 98–99
- FindString function, 36
- FindStringExact function, 36
- FirstDayOfWeek property, 37
- FlatStyle property, 31
- Focus function, 31, 32
- folder locations, determining, 510
- Font property, 30, 32
- fonts
 - listing and setting, 485
 - OpenType, 485
 - TrueType, 485
- Footer, Page, 224, 239
- Footer, Report, 224, 239
- For-Next loop, 69
- ForeColor property, 30, 353
- foreign keys, 143, 163
- Form control, 287
- Form control (mobile), 324
- form errors, highlighting, 499–500
- form grid, snapping to, 478
- Form Icon property, 109
- form-level variable, declaring, 54–56
- Format function, 489
- forms
 - adding multiple, 77–80
 - Launch Calculator menu item, 78
 - modules, 80
 - As New statement, 80
 - preventing unwanted instances, 79–80
 - Windows Form, 77
 - adding notes to, 362
 - introduction to, 8
 - multiple, 76
 - resetting, 503–504
 - transparent, 478
- Forms authentication, 294, 317
- FormsAuthentication class, 298
- Friend prefix, 405
- FrontPage, designing interfaces with, 304–305
- FTP, 310, 318, 438
- functions
 - code sample, 93
 - defined, 93
 - expanding program with, 83–88
 - Add function, 86–87
 - code samples, 84
 - Label control, 86

- functions (*continued*)
 - login with parameters (sample code), 85
 - number-adding program, 85–88
 - methods and, 26–27, 49
- G**
 - GAC (global assembly cache), 411, 470
 - garbage collection, 465
 - Garbage Collection class, 408
 - GeneratePassword function, 519
 - Get and Set blocks, 389, 458
 - Get code block, 395
 - GIF images, 354, 365
 - global methods and functions, 91
 - Global.asax files, 246, 313
 - greater-than (>) symbol, 46
 - GroupBox control, 34
 - GroupName property, 268
 - GUIDs, creating, 507–508
- H**
 - Handles keyword, 407, 467, 497
 - Handles statements, 390
 - HasBackButton device property, 347
 - Hashtable class, 529
 - Header, Report, 224, 239
 - Helper class code, 427
 - Hide function, 31, 34
 - HTML
 - 3.2 (mobile sites), 341
 - code
 - mobile applications and, 354, 361
 - Web design and, 304
 - controls, 247
 - elements, 284
 - source, 253
 - tables, 265
 - Hyperlink control, 259–260, 278, 312
- I**
 - icons
 - changing form, 109
 - creating, 495
 - ID property, 251–252
 - Identity Seed and Increment properties, 195
 - If statements, 44–46
 - If-Then statements, 108, 108–109
 - IIF function, 500
 - Image control, 269–270
 - Image property, 31, 35
 - ImageAlign property, 31
 - ImageButton control, 270–271
 - ImageIndex property, 491
 - ImageKey property (mobile), 338
 - ImageList control, 491
 - images, displaying (mobile applications)
 - basics, 354
 - mobile device formats, 334
 - using filters, 354–358
 - Images property, 491
 - ImageURL property, 269
 - ImageURL property (mobile), 334
 - Immediate mode, 490
 - IndexOrKey parameters, 395
 - inheritance, 371, 409, 462, 469–470
 - InitializeComponent method, 468
 - InputBox function, 53
 - Insert statements (SQL), 168, 187
 - int (integer), 195
 - integer upgrades, 456–457
 - Intellisense, 87, 466
 - interactive Web sites. *See* ASP.NET Web applications
 - Interface keyword, 471
 - interfaces, 304–305, 471
 - Intermediate Language, 471
 - Internet Explorer, 245, 253, 309, 481
 - Internet Explorer WebControls
 - expansion pack, 312–313
 - Internet Information Server 5.0, 244
 - Internet Information Services (IIS), 294, 317, 533
 - Internet Services Manager
 - plug-in, 437
 - invisible controls, 49
 - IP Addresses, 283
 - IsColor device property, 347
 - IsDate function, 64
 - IsEmailAddress function, 501
 - isEricssonR380 filter, 353
 - isHTML32 filter, 349
 - IsLeapYear function, 507
 - IsMobileDevice device property, 347
 - isNokia7110 filter, 353
 - IsNumeric function, 64, 88, 210
 - IsPostBack property, 283
 - IsValid property, 264
 - IsWholeNumber function, 509
 - ItemCommand event (mobile), 335
 - Items collection (mobile), 335
 - Items property, 36, 260
- J**
 - Jet MDB database, 163, 233
 - Join function (arrays), 526
- K**
 - KeyDown events, 32
 - KeyPress event, 32
 - KeyUp event, 32

KeywordFilter property, 269, 306
 KeywordFilter property (mobile), 338
 keywords as variable names, 506
 keywords (listed)
 With, 479
 #Region and #End Region, 454
 AddressOf, 466
 ByVal, 85
 Class, 462
 Default, 463
 Handles, 407, 467, 497
 Interface, 471
 KeywordFilter, 269
 KeywordFilter property, 306
 KeywordFilter property
 (mobile), 338
 Namespace, 461
 Optional keyword default, 407
 Overrides, 470
 Private and Public, 89–90, 390
 ReadOnly, 395, 463
 Return, 458
 SyncLock, 314
 TODO, 480
 WithEvents, 385, 390, 406
 WriteOnly, 395, 463

L

Label control, 30, 86
 Launch Calculator
 menu item, 78
 layers, application, 403
 LayoutMdi method, 491
 LCase function, 486, 498
 leap year, checking for, 507
 Leave event, 32
 less-than (<) symbol, 46
 Link control (mobile), 324
 LinkBehavior property, 31
 LinkButton control, 259–260
 LinkClicked event, 31
 LinkColor property, 31
 LinkLabel control, 30–31
 links, opening new, 312
 list type controls, 260–261
 ListBox control, 35–36, 183, 260–261,
 312, 503
 Location box, defined, 7
 login with parameters
 (sample code), 85
 loops
 computer, 65–67, 69
 declaring variables inside, 459
 Do-Until, 70
 example, 399
 For-Next, 69

M

macros, 514
 MainMenu control, 72–73, 93, 479
 managed code, 110
 master forms (MDI), 491
 mathematical operators/functions, 88,
 152
 MaxDate property, 37
 MaxDropDownItems property, 36
 MaximumValue property, 263
 MaxLength property, 32
 MaxLength property (mobile), 332
 MDI (Multiple Document Interface)
 applications, 490–491
 MdiContainer property, 491
 MdiList property, 76
 MdiParent property, 491
 memory (mobile applications), 358–360
 MenuComplete event, 76
 MenuItem control, 72–75, 76, 93
 MenuItem entries, 479
 menus
 adding, 71–76
 MainMenu control, 72–73
 MdiList property, 76
 MenuComplete event, 76
 MenuItem control, 72–75, 76
 MenuStart event, 76
 Shortcut property, 74
 to Windows forms, 479
 graphic design of, 93
 MenuStart event, 76
 message boxes, displaying, 12–16
 Message Queuing Services, 534
 MessageBox function, 59–62
 MessageBox.Show function, 59, 61
 metadata, 21, 469
 methods
 .Show method (forms), 93
 adding, 81–83
 adding descriptions to, 436
 creating (“Goodbye” subroutine),
 81–83
 defined, 93, 382
 Dispose method, 408
 Exit method, 486
 Finalize method, 386–387, 408,
 464–465
 and functions, 26–27, 49
 InitializeComponent method, 468
 LayoutMdi method, 491
 New method, 386, 390
 parameters and, 94
 RaiseEvent method, 381, 390
 Response.Redirect method, 360
 shared, 499

- Microsoft Access, 122
 - Microsoft Access database, creating, 130–145
 - field names, 138
 - name table, adding, 141
 - primary keys, 139
 - relationships, adding, 141–145
 - Microsoft and System namespaces, 57–59
 - Microsoft Desktop Engine (MSDE), 190–191
 - Microsoft emulator tests (Web site), 362
 - Microsoft Intermediate Language (MSIL) code, 17–18, 471
 - Microsoft Mobile Explorer Emulator, 323, 328–329
 - Microsoft UDDI Directory, 430, 443
 - Microsoft Visual Basic namespace, 58, 69
 - Microsoft Visual Studio.NET. *See* Visual Studio .NET
 - Microsoft Web controls (new), 313
 - mini scripts, 236
 - MinimumValue property, 263
 - mobile devices
 - adaptor plug-ins for, 361
 - determining target devices, 343–348
 - Device object, 344
 - important device properties, 346–348
 - Mobile Internet Toolkit (MIT), 321–342
 - background and basics, 321–322
 - creating mobile Web applications, 322–330
 - adding code, 327–328
 - mobile Web forms, 323–326
 - testing application, 328–330
 - download location, 323
 - Mobile Web Form controls, 330–339
 - AdRotator control, 338
 - Calendar control, 338
 - Command control, 333
 - DeviceSpecific control, 337
 - Form control, 330
 - Image control, 334
 - Label control, 331
 - Link control, 333
 - List control, 335
 - Mobile QuickStart tutorial, 330, 336
 - ObjectList control, 336
 - Panel control, 331
 - PhoneCall control, 334
 - SelectionList control, 335–336
 - StyleSheet control, 337
 - TextBox control, 332
 - TextView control, 332
 - Validation controls, 339
 - Mobile Web Form controls. *See* under Mobile Internet Toolkit (MIT)
 - MobilePages, 330
 - modules
 - adding to program, 89–91
 - classes inside of, 470
 - defined, 89, 94
 - form instances and, 80
 - functionality through class libraries, 405–406
 - public and private, 94
 - MonthCalendar control, 37–38, 503
 - MouseMove event, 30, 34
 - MSMQ (Microsoft Message Queue), 528
 - MTS and MSMQ, 528
 - MultiColumn property, 36
 - MultiLine property, 32
 - multiple If statements, 46
- ## N
- n-tier applications, 403
 - Name properties, 24–25
 - Namespace keyword, 461
 - namespaces, 57–59, 69, 437, 460–461, 470
 - naming conventions (prefixes), 25, 542–544
 - NavigateUrl property, 353
 - NavigateUrl property (mobile), 333
 - .NET
 - background and components, 447–450
 - code, 110
 - hosting accounts, 310
 - objects and COM, 513–514
 - .NET Framework
 - client browser and, 275
 - components (controls), 9
 - defined, 4–5, 21, 449–450
 - New method, 386, 390
 - Nokia Mobile Internet Toolkit, 362
 - non-deterministic finalization, 465
 - not-equal-to (<>) symbol, 46
 - number-adding program, 85–88
 - number-only text box, 483
 - numbers
 - retrieving using Val, 486
 - rounding, 484
 - numeric data types, 457
 - Numeric property (mobile), 325, 332
 - NumericUpDown control, 503

O

Object data type, 457
objects, 369–390, 462–471
 adding namespaces, 470
 ASP.NET, 438
 attributes, 468–469
 Class keyword, 462
 class libraries, 462
 constructors, 464
 creating components, 471
 default properties, 463
 delegates, 466–467
 inheritance, 469–470
 interfaces, 471
 overloading, 465–466
 property changes, 463
 reference error message, 234
 setting to Nothing, 373
 shared members, 467–468
 side-by-side execution
 (DLLs), 470
 termination, 464–465
 using modules, 470
 in VB .NET, 109
 WithEvents keyword and, 406
objects and classes
 background, 369
 benefits of, 370–371, 389
 classes and objects, creating, 372–376
 class naming conventions, 372
 multiple class instances, 374
 defined, 369–370, 389
 enumerations, adding, 378–381
 events to class, adding, 383–385
 Finalize method, 386–387, 390
 methods/functions/parameters,
 adding, 382–383
 properties to class, adding, 376–378
objects (listed)
 Application object, 278, 313, 529
 ArrayList object, 528
 BitArray object, 528
 Clipboard object, 498
 Collection object, 395
 Device object, 344, 365
 OleDbConnection object, 129,
 174–175
 OleDbDataAdapter object, 129–130,
 163
 Page object (defined), 278
 Request object, 278, 279, 281, 283
 Response object, 278
 Server object, 278
 Session object, 278, 280, 288, 359
 User object, 278, 317
OleDb namespace, 204

OleDb strings, generating, 508
OleDbConnection, 163
OleDbConnection object, 129, 174–175
OleDbDataAdapter object, 129–130, 163
One-to-Many relationship type, 144
OOP (object-oriented programming)
 defined, 369, 389
 five key rules of, 409, 462
 multiple layers and, 403
Opacity property, 478
OpenType fonts, 485
OpenWave Mobile Browser, 362
Option Strict setting, 535
Optional keyword default, 407
overloading (multiple constructors),
 464–466
Overrides keyword, 470

P

Page Footer, 224, 239
Page object (defined), 278
PageSettings class, 521
pagination (mobile Web), 332
parameters (arguments)
 apostrophes and, 235
 defined, 14
 “e” parameter, 491
 IndexOrKey, 395
 login function with, 84–86
 in methods and functions, 94
 to simplify code, 210
 storing user values in registry, 83
parent-to-child relationships, 163
Passport authentication, 294, 317
password program, creating, 56–57
Password property (mobile), 332
PasswordChar property, 32
passwordFormat attribute, 296
passwords, generating memorable,
 519–520
PhoneNumber property (mobile), 334
Picture Viewer application, creating,
 38–44
 adding code, 41–42
 testing program, 42–44
PictureBox control, 35, 40
pipe symbols, 41
polymorphism (defined), 408–409, 462
Postbacks, 283, 529
PreferredRenderingType property, 349
prefersGif filter, 365
prefersWBMP filter, 365
prefixes
 for controls and variables, 542–544
 naming, 25
 three-character, 69

- presentation layers, 403
- PrevInstance function, 510
- previous instances, checking, 510–511
- primary keys, 139, 142–143, 163
- PrintDocument class, 521
- PrinterSettings class, 521
- printing from programs, 520–523
- PrintPreviewControl class, 521
- Private and Public keywords, 89–90, 390
- programs
 - compiling (basics), 17–18
 - creating first, 5–12
- project defaults (Visual Studio .NET), 536–541
 - class libraries, 537–538
 - Web applications, 538–539
 - Web services, 539–541
 - Windows applications, 536–537
- properties
 - adding to classes, 376–378
 - changes, 463
 - declaring, 458
 - defined, 49
 - Get and Set blocks, 376, 389
 - mobile device, 346–348
 - reading/writing in code, 24–27
 - setting and reading, 26
- properties (listed)
 - ActiveForm property, 491
 - ActiveLinkColor property, 31
 - AdvertisementFile property, 269
 - Anchor property, 40
 - Argument property, 353
 - Browser device property, 347
 - Calendar properties, 254–255
 - CanInitiateVoiceCall device property, 347, 349
 - CanSendMail device property, 347
 - CauseValidation property, 264
 - Checked property, 33
 - CheckState property, 32
 - ControlToValidate property, 263
 - Cookies device property, 347
 - Cursor property, 31, 482
 - DataSource property, 265
 - Display property, 263
 - Dock property (controls), 479
 - Document property, 272
 - DocumentContent property, 272
 - DocumentSource property, 272
 - DrawGrid property, 478
 - DropDownStyle property, 36
 - EnableClientScript property, 264
 - EnableViewState property, 288, 362
 - FirstDayOfWeek property, 37
 - FlatStyle property, 31
 - Font property, 30, 32
 - ForeColor property, 30, 353
 - Form Icon property, 109
 - GroupName property, 268
 - HasBackButton device property, 347
 - ID property, 251–252
 - Image property, 31, 35
 - ImageAlign property, 31
 - ImageIndex property, 491
 - Images property, 491
 - ImageURL property, 269
 - IsColor device property, 347
 - IsMobileDevice device property, 347
 - IsPostBack property, 283
 - IsValid property, 264
 - Items property, 36, 260
 - KeywordFilter property, 269, 306
 - LinkBehavior property, 31
 - LinkColor property, 31
 - MainMenu control, 479
 - MaxDate property, 37
 - MaxDropDownItems property, 36
 - MaximumValue property, 263
 - MaxLength property, 32
 - MdiContainer property, 491
 - MdiParent property, 491
 - MinimumValue property, 263
 - Mobile Web application
 - properties
 - ActiveForm property, 324
 - AdvertisementFile property, 338
 - AlternateFormat property (mobile phone), 334
 - AlternateText property, 334
 - CalendarEntryText property, 338
 - ImageKey property, 338
 - ImageURL property, 334
 - KeywordFilter property, 338
 - MaxLength property, 332
 - NavigateUrl property, 333, 334
 - NavigateUrlKey property, 338
 - Numeric property, 332
 - Password property, 332
 - PhoneNumber property, 334
 - ReferencePath property, 337
 - SelectedDate property, 338
 - Selection property, 335
 - SelectionMode property, 338
 - SelectType property, 335
 - StyleReference property, 328, 331, 337
 - Text property (mobile phone), 334
 - VisibleDate property, 338
 - MultiColumn property, 36
 - MultiLine property, 32

- NavigateUrl property, 353
- Opacity property, 478
- PasswordChar property, 32
- PreferredRenderingType property, 349
- PropertyOverrides property, 365
- QueryString property, 286
- ScreenCharactersHeight device property, 347
- ScreenCharactersWidth device property, 347
- ScreenPixelsHeight/Width device property, 347
- ScrollBars property, 32
- Selected Text property, 36
- SelectedIndex property, 261
- SelectedItem property, 267–268, 312
- SelectionMode property, 36
- SelectionRange property, 37
- ShowToday property, 37
- ShowTodayCircle property, 37
- ShowWeekNumbers property, 37
- smartNavigation property, 309, 317–318, 481
- SnapToGrid property, 478
- Sorted property, 36
- StartPosition property, 479
- SupportsBodyColor device property, 347
- Supports(text styles) device property, 347
- TabIndex property, 481
- TabStop property, 481
- Tag property, 503
- Target property, 312
- Text property, 30–34, 36, 491
- TextAlign property, 30, 31, 32
- ToolTipText property, 491
- TransformSource property, 272
- Type property, 263
- UserHostAddress/UserHostName, 283
- Visible property, 34
- WindowState property, 479
- Properties window, 10, 21
- PropertyOverrides property, 365
- Public Class Form1, defined, 12
- public variables, 90, 94

Q

- query strings
 - coding, 285–287
 - storing user data, 359–360
- Queue object, 528
- quiz, 288–289
- quotation marks, use of, 14

R

- RadioButton control, 33–34, 268–269, 503
- RadioButtonList control, 268–269
- RaiseEvent, 467
- RaiseEvent method, 381, 390
- Random class, 65–66
- random numbers, generating, 481–482
- RangeValidator control, 263
- read-only properties, 27
- ReadOnly keyword, 395, 463
- records (defined), 233
- Recordset into DataSet, 513
- RedirectFromLoginPage (authentication), 317
- RedirectToMobilePage function, 341
- reference books
 - building mobile Web applications, 363
 - building Web applications (ASP.NET), 314
 - database reference, 236
 - OOP (object-oriented programming), 411–412
 - VB .NET and .NET platform, 473
 - VB .NET reference, 112
 - Web services, 440
- reference data types (defined), 549
- ReferencePath property (mobile), 337
- REGEDIT.EXE, 511
- #Region and #End Region keywords, 13, 454
- registry, saving, 511–512
- Registry screen, defined, 107
- RegularExpressionValidator control, 263, 500–501
- relationships
 - database, 141–145, 163
 - in reports, 225
 - SQL Server database, 197–200
- Repeater control, 265–266
- Replace() function, 210, 484
- reports, designing, 219–232
 - adding data to report, 227–229
 - creating with Crystal Report, 220–224
 - grouping data, 230–232
 - options for creating, 220
 - storing as RPT files, 239
- Request object, 278, 279, 281, 283
- RequiredFieldValidator control, 262–264, 326
- Response object, 278
- Response.Redirect method, 360
- Resume Next statements, 96–98, 115
- retrieving numbers (Val), 486

- Return statement, 458, 482
- Reverse operations (Web services), 425, 428
- reversing strings, 484
- Round function, 484
- rounding numbers, 484
- RPT files (reports), 239
- runtime
 - defined, 11
 - errors, 96–100
- S**
- SaveFileDialog control, 63
- ScreenCharactersHeight device property, 347
- ScreenCharactersWidth device property, 347
- ScreenPixelsHeight/Width device property, 347
- ScrollBars property, 32
- Secure Sockets Layer (SSL), 312
- security. *See* authentication
- Select Case statements (example), 381
- Select statements (SQL), 168, 187
- Select @@Identity statement, 214, 218
- SelectedDate property (mobile), 338
- SelectedIndex property, 261
- SelectedIndexChanged event, 37, 261, 312
- SelectedItem property, 267–268, 312
- Selection property (mobile), 335
- SelectionMode property, 36
- SelectionMode property (mobile), 338
- SelectType property (mobile), 335
- Server Explorer, 192
- Server object, 278
- server-side controls, 529
- Service classes, multiple (Web services), 438
- Session object, 278, 280, 288, 359
- Set block, 395
- Set statements, 459–460
- SetDate event, 37
- SetSelectionRange event, 37
- Setup Bootstrapper, 111
- setup .EXE file, 110–111
- setup programs, building, 100–108
 - adding shortcut to Programs menu, 104–105
 - adding subfolder to Programs menu, 105
 - Bootstrapper option, 107
 - customizing setup, 107–108
 - Primary Output file, adding, 103–104
 - review and summary, 115–116
 - setting Properties, 107
 - setup .EXE file, 110–111
 - testing setup program, 105–106
- Setup Project, 115
- shared members, 467–468
- shared methods, 499
- Short (data type), defined, 54–55, 69
- Short variables, 376
- shortcut keys
 - F1 (help), 42
 - F2, 59
 - F5 (run applications), 42, 443
 - F7, 423
 - F8, 44
 - F9, 43, 67
- Shortcut property, 74
- Show function, 31, 34, 467
- .Show method (forms), 93
- side-by-side execution, 411, 470
- smallmoney (data type), 195
- smartNavigation property, 309, 317–318, 481
- SnapToGrid property, 478
- SOAP (Simple Object Access Protocol), 421, 452
- Solution Explorer, 15–16, 246, 453
- Split function (arrays), 526
- SQL, asking questions in
 - Access-generated SQL statements, 170–171
 - Datasheet view, 167–168
 - SQL statements, 168–169
 - SQL view, 167
- SQL Select statements, 233
- SQL Server, 58, 189–218
 - creating SQL Server database, 191–200
 - adding relationships, 197–200
 - designing database tables, 194–197
 - linking tables, 198–199
 - Server Explorer, 192
 - creating Web application database, 200–215
 - adding data, 206–210
 - adding transactions, 210–215
 - Command and DataReader objects, 206
 - designing Web page, 201–202
 - setting up connection objects, 203–204
 - writing the code, 202–206
 - definition and background, 190–191
- SQL Server data types (listed), 550–551
- SQL Server database, 191–200
 - adding relationships, 197–200
 - designing database tables, 194–197

- linking tables, 198–199
 - Server Explorer, 192
- SQL statements, 165–169, 187, 235
- SqlClient namespace, 203–204
- Stack object, 528
- Start button (F5), 16
- Start function launching routine, 61–62
- StartPosition property, 479
- static control, 263
- static HTML, 529
- Static keyword, 507
- static (shared) members, 467
- static variables, declaring, 507
- stored procedures, 236
- StrConv function, 498
- StreamWriter object, 62
- String (data type), defined, 55, 69
- StringBuilder class, 529, 549
- strings
 - filtering, 41
 - language changes, 457
 - reversing, 484
 - setting, 26
 - trimming, 485
- StrReverse function, 484
- Structure, 458–459
- StyleReference property (mobile), 325–326, 328, 331, 337
- styles, defining and using, 307–309
- stylesheets. *See* cascading style sheets
- subroutines
 - adding to structures, 459
 - defined, 13
 - events and, 49
 - generated automatically, 28
- symbols. *See also* characters
 - & concatenation character, 41
 - + for query strings, 286
 - + operator, 152
 - greater-than (>), 46
 - less-than (<), 46
 - not-equal-to (<>), 46
 - pipe, 41
- SyncLock keyword, 314
- System namespace, 58–59, 69
- System.Data.OleDb namespace, 217
- System.Data.SqlClient namespace, 217
- System.Math namespace, 88
- System.Random class, 481
- System.Web.Security namespace, 298

T

- tab order, changing, 481
- TabIndex property, 481
- tables
 - in Access, 163

- database (defined), 121
- SQL Server database, 194–197, 198–199
- TabStop property, 481
- Tag property, 503
- Target property, 312
- termination, 464–465
- text
 - converting case, 498
 - easy replacement, 484–485
- Text property, 30–34, 36, 491
- Text property (mobile phone), 334
- text string variables, creating, 52–53
- text strings, 14
- TextAlign property, 30–32
- TextBox control, 25–26, 32, 503
- TextChanged event, 32, 37
- timestamp (data type), 195
- tips and techniques
 - advanced, 505–529
 - .NET objects in COM world, 513–514
 - arrays, creating and sorting, 523–525
 - arrays, using special functions, 526–527
 - ASP.NET applications, fine-tuning, 529
 - checking for leap year, 507
 - collections in .NET, 527–528
 - COM objects, using, 512
 - command line parameters, reading, 509–510
 - declaring static variables, 507
 - encrypting files, 516–519
 - encryption, using simple, 514–515
 - folder locations, determining, 510
 - GUIDs, creating, 507–508
 - keywords as variable names, 506
 - macros, saving time with, 514
 - MTS and MSMQ, finding, 528
 - OLE DB strings, generating, 508
 - passwords, generating memorable, 519–520
 - previous instances, checking for, 510–511
 - printing from programs, 520–523
 - Recordset into DataSet, 513
 - registry, saving to, 511–512
 - storing often-used code, 514
 - VS .NET command prompt, 506
 - whole numbers, checking for, 509
 - beginner, 477–486
 - capitalizing characters, 486
 - controls, resizing at runtime, 478–479

- tips and techniques (*continued*)
 - cursors, customizing, 482
 - ending applications, 486
 - exiting code, 482
 - fonts, listing and setting, 485
 - form grid, snapping to, 478
 - forms, making transparent, 478
 - With keyword, adding, 479
 - menus to programs, adding, 479
 - number-only text box, creating, 483
 - numbers, retrieving using Val, 486
 - random numbers, generating, 481–482
 - reversing strings, 484
 - rounding numbers, 484
 - site homepage, setting, 480–481
 - Smart Navigation to Web forms, adding, 481
 - special code characters, 482–483
 - tab order, changing, 481
 - text replacement, 484–485
 - trimming strings, 485
 - typing TODO: statement, 480
 - visual inheritance, 480
 - Windows forms, changing state of, 479
 - windows, resetting, 486
 - intermediate, 487–504
 - Application directory, finding, 488
 - clipboard, storing to, 498
 - Command window, exploring, 490
 - DataSets and XML, using, 494
 - dates, formatting, 488–489
 - e-mail addresses, validating, 500–503
 - files, manipulating, 498–499
 - files, reading and writing to, 495–496
 - form errors, highlighting, 499–500
 - forms, resetting, 503–504
 - Handles keyword, using, 497
 - icons, creating, 495
 - IIF function, using, 500
 - last day of month, finding, 496–497
 - MDI applications, creating, 490–491
 - text case, converting, 498
 - toolbars, adding, 491–492
 - XML files, reading, 492–494
 - TODO keyword, 480
 - ToLower and ToUpper properties, 45
 - ToolBar control, 491
 - toolbars, adding, 491–492
 - toolbox (basics), 9
 - ToolTipText property, 491
 - transactional database-powered Web application, 200–210
 - transactions
 - adding to Web application database, 210–215
 - defined, 218
 - TransformSource property, 272
 - Trim function, 485
 - trimming strings, 485
 - TrueType fonts, 485
 - Try blocks, 98
 - Try-Catch-Finally blocks, 98–100, 115, 460
 - Type Library Importer tool, 512
 - Type property, 263
 - Typed/Untyped DataSets, 147–149, 430
- ## U
- UCase function, 486, 498
 - underscore character, 60, 482–483
 - Undo function, 32
 - uniqueidentifier (data type), 195
 - unmanaged code, 110
 - untyped DataSet Control, 131
 - untyped DataSets, 147
 - Update statements (SQL), 168, 187
 - Update Web Reference, 439
 - UpdateDB method, 402
 - updating database, 135, 152–153
 - Upgrade Wizard, 472
 - upgrading existing applications, 472
 - uploading files (coding), 284–285
 - URLs and Refferers, 282
 - URLs (Uniform Resource Locators), 282
 - user-defined types, 458–459
 - User Interface screen, defined, 107–108
 - User object, 278, 317
 - UserHostAddress/UserHostName properties, 283
 - users
 - code for redirecting, 278–279
 - code for remembering sessions, 280
 - coding for grabbing IP addresses, 283
 - logging browsers of, 281–282
- ## V
- Val function, 64, 88
 - Validation controls, 261–264
 - ValidationSummary control, 264
 - value data types, 549
 - varchar (data type), 195
 - variable names, keywords as, 506
 - variables, 51–56
 - creating text string variable, 52–53
 - declaring form-level variable, 54–56

- declaring static, 507
 - defined, 52, 69
 - parameters and, 85
 - prefixes for, 542–544
 - public, 90
 - setting value of, 69
 - for storing data in objects, 376
 - three-character prefixes, 69
- Variant and Currency, 457
- VB .NET
 - background and history, 4–5
 - compiling and distributing applications, 471–472
 - data types (listed), 548–549
 - language changes, 456–461
 - array alterations, 458
 - block-level scoping, 459
 - ByVal default, 459
 - declaring properties, 458
 - error-handling changes, 460
 - integer upgrades, 456–457
 - namespaces, 460–461
 - no Set statement, 459–460
 - strings, 457
 - user-defined types, 458–459
 - Variant and Currency eliminated, 457
- objects, 462–471
 - adding namespaces, 470
 - attributes, 468–469
 - Class keyword, 462
 - class libraries, 462
 - constructors, 464
 - creating components, 471
 - default properties, 463
 - delegates, 466–467
 - inheritance, 469–470
 - interfaces, 471
 - overloading, 465–466
 - property changes, 463
 - shared members, 467–468
 - side-by-side execution (DLLs), 470
 - termination, 464–465
 - using modules, 470
- upgrading existing applications, 472
- VB6 Variant, 109
- _VIEWSTATE field, 253, 275, 287–288, 359, 361–362
- Visible property, 34
- Visual Basic, 5, 21
- Visual C#, 5
- Visual C# .NET, 450
- Visual C++, 5
- Visual C++ .NET, 450
- visual inheritance, 480
- Visual SourceSafe, 533
- Visual Studio .NET, 450–456
 - command prompt, 506
 - creating Windows application, 452–456
 - defined, 5
 - exploring basics, 450–452
 - installing, 533–535
 - project defaults, 536–541
- VSDISCO files, 246
- W
- WAP (Wireless Application Protocol), 323, 341, 361
- WBMP (wireless bitmap) images, 354–355, 365
- Web application database, 200–215
 - adding data, 206–210
 - adding transactions, 210–215
 - Command and DataReader objects, 206
 - designing Web page, 201–202
 - setting up connection objects, 203–204
 - writing the code, 202–206
- Web applications
 - creating, 244–246, 248–258
 - project defaults for, 538–539
 - reference books/Web sites (ASP.NET), 314
- Web Bootstrapper, 110
- Web Form controls, 247
- Web forms, 164, 249–250, 312
- Web pages, designing database, 201–202
- Web services, 419–444, 437–438
 - accessing from applications, 425–429
 - accessing from browsers, 424–425
 - ASP.NET objects, 438
 - CacheDuration, 439
 - changing servers, 440
 - creating Web services, 422–424
 - debugging, 438
 - deploying Web service, 437–438
 - distributing to servers, 444
 - distribution tips, 435–437
 - exposing code, 443
 - fundamentals, 419–422
 - possible uses for, 429–430
 - project defaults for, 539–541
 - security and authentication, 439
 - server and client sides, 420–421, 435, 443
 - smart data, 430–435
 - passing smart data demo, 430–433
 - retrieving smart data, 434–435
- SOAP (Simple Object Access Protocol), 421

- Web services (*continued*)
 - Update Web Reference, 439
 - XML (eXtensible Markup Language), 421
 - Web sites (for further reference)
 - Ad agency, 305
 - CSS tutorial, 309
 - database info, 236–237
 - Developer Fusion (icons), 109
 - free image library, 355
 - Microsoft emulator tests, 362
 - Microsoft Mobile Explorer Emulator
 - download, 323
 - Microsoft new Web controls, 313
 - Microsoft UDDI Directory, 430, 443
 - Mobile Internet Toolkit download, 323
 - mobile Web applications, 363
 - Nokia Mobile Internet Toolkit download, 362
 - OOP (object-oriented programming), 411–412
 - OpenWave Mobile Browser download, 362
 - passports, 294
 - setting homepage, 480–481
 - VB .NET and .NET platform, 473–474
 - VB .NET programming introduction, 112–113
 - Web applications (ASP.NET), 314–315
 - Web services, 440–441
 - Web sites, interactive. *See* ASP.NET Web applications
 - Web.config file
 - editing, 296
 - filters in, 365
 - mobile projects, 349
 - options, 317
 - and security, 311
 - settings storage, 246
 - WebMethod attribute, 439
 - whole numbers, checking, 509
 - Windows 2000 and XP, 534
 - Windows applications
 - creating in Visual Studio .NET, 452–456
 - project defaults for, 536–537
 - Windows authentication, 294, 317
 - Windows Form, 77
 - Windows Form Designer generated code, 12
 - Windows forms
 - changing state of, 479
 - data binding to, 145–154
 - binding text boxes to data, 150–152
 - deleting records, 154
 - moving around rows, 152
 - typed and untyped DataSets, 147–149
 - updating database, 152–153
 - default controls (listed), 545–547
 - Windows Installer, 105
 - Windows Installer Bootstrapper, 107
 - Windows Installer MSI package, 110–111
 - windows, resetting, 486
 - WindowState property, 479
 - Wireless markup Language (WML), 323
 - With keyword, 479
 - WithEvents, 467
 - WithEvents keyword, 385, 390, 406
 - Wizards, Data Form, 159–160
 - WML (Wireless Markup Language), 323, 341, 354, 361
 - WriteOnly keyword, 395, 463
 - WSDL (Web Service Discovery Language) files, 427
- ## X
- XCOPY deployment, 100, 310, 472
 - XML (eXtensible Markup Language)
 - .NET and, 425
 - advertisement file, 269, 338
 - control, 272
 - DataSets and, 494
 - document example, 234
 - objects in System.Xml, 58
 - reading files, 492–494
 - Web services and, 421, 443
 - XMLDocument object, 272
 - XSL (Extensible Stylesheet language) files, 272