CHAPTER 1

Data Sources and the Web

Look around you. No really, look around you. In the past 30 years, computers have taken over from the filing cabinets of the world to become the (almost) universal way people store and look up information. Would you rather spend five minutes rifling through some badly organized stack of paper for the name of a client or the price of a book, or spend ten seconds typing in a search query on a computer and getting the desired information back immediately? I thought so—the computer wins every time.

It's not just in the office that data-driven Web sites have proven popular. Server-side technologies now allow people to hook electronic data sources—databases, spreadsheets, Extensible Markup Language (XML) files, Windows services, and more—to Web sites. This means that today's World Wide Web is a place of dynamic, data-driven Web sites, rather than the collections of static Hypertext Markup Language (HTML) pages it once was. Regardless of whether you develop your Web sites with Active Server Pages (ASP), ASP.NET, PHP, JavaServer Pages (JSP), or one of numerous other technologies, you can use a data source to interact with your users, giving them the information they want to see and safely storing how they want to see it next time.

E-commerce Web sites, such as Amazon and eBay, use databases to provide customers with product information, recommendations, and wish lists, and to store feedback and orders. Portal Web sites use databases to store articles and user settings, so users don't need to reset them each time they visit the Web site.

How you choose to use data in your Web site is up to you. Whatever your goals are for your data-driven Web site, this book will give you the tools you'll need to accomplish them.

In this chapter, you'll look at the world of data-driven Web sites from 50,000 feet, so that by the time you finish it, you'll at least have a rough knowledge of how things hook together. You'll spend the rest of the book parachuting down to the ground, espying the exact details as you get closer.

Up here in the blue sky of Chapter 1, you'll learn the following:

- Why data-driven Web sites are such a good idea
- How a data-driven page actually works
- The different sources of data you can use with ASP.NET Web sites
- How ADO.NET is the glue that joins data sources and ASP.NET Web sites
- How to build your first data-driven page

If you've already read a beginner's book on ASP.NET, such as *Beginning ASP.NET 2.0 in C#2005: From Novice to Professional* by Matthew MacDonald (Apress, 2006), you're probably familiar with some of the material in this chapter already, so you could skip to the next chapter. Still, I encourage you at least to browse through this chapter. You never know what nuggets of information you may find.

Are Data-Driven Web Sites a Good Idea?

Should you even bother with hooking a data source to your Web site? That's the $64,000 question, isn't it really? If you're reading this book, I'll assume you've already come to the conclusion that using databases and other sources of data to turn static Web sites into dynamic data-driven Web sites is a good thing. However, I would be lying if I said there weren't any disadvantages to using data sources—there are. This section, then, covers the pros and cons of creating data-driven Web sites.

On the plus side, data-driven Web sites offer the following:

Maintenance: Using a database makes it a lot easier to maintain your data and keep it up-to-date. Take the example of a bank application that contains lists of customers by name and by branch, and contains profiles for each customer. Each time the customer is mentioned in a list, the customer's account number is also present. If that account number changed, the application would need to change it accordingly on all the lists, which could lead to errors; after all, account numbers aren't the easiest things to remember. A well-designed database usually ensures that easily mistyped data—such as Social Security numbers (SSNs), credit card numbers, International Standard Book Numbers (ISBNs), and so on—is entered or modified in only one place, rather than several. The data-driven Web site would then generate the lists by querying the database. Another reason that data-driven Web sites are easier to maintain is that they typically have fewer actual pages than static Web sites. The pages they do have act as templates that are filled on-the-fly from a database, as opposed to the complete, individual pages that static Web sites contain.

Reusability: Information in databases can easily be backed up and reused elsewhere as required. Compare this to static Web sites, where the information can't be retrieved easily from the surrounding HTML and layout instructions.

Data context: Databases allow you to define relationships and rules for the data in your database. For example, you can create a rule in your database that says if you store some information about a book, you must include an author and an ISBN, which must, in turn, be valid. This means that rather than querying the database for information by a simple index, such as the one in the back of this book, you can specify what to search for, as well as the order in which the information should be returned. A great example of this is the search engine. Can you imagine Google as a table of contents for the Web?

Quality and timeliness of content: Databases are optimized for the storage and retrieval of data and nothing else. They allow you to use and update information on a live Web site almost in real time—something that isn't possible with a Web site consisting of just static pages containing forms. For example, consider what happens when an e-commerce Web site receives an order for some goods. The code running behind the page knows to store the new order in a database and to reduce the inventory count for each item in the order once payment has been received. If the customer wants to change the order, it's still available in the database to be changed. The inventory also can be changed, depending on what the customer does. For instance, if the customer cancels the order, the system can simply reinstate inventory levels and mark the order as canceled. Now consider what happens if the e-commerce Web site has a human on the other side instead of a database, and the customer wants to change the order. The human needs to find the order, check the stock, and so on. This process wouldn't be immediate, and it would be prone to errors. What if the order were lost or incorrectly recorded?

On the downside, data-driven Web sites have some additional requirements:

Development time: It takes a little more time to write code to access the database containing information and to populate the database with the information you require. Likewise, it may take a little more planning initially to accommodate a database in the architecture of a Web site. Sometimes, the data may not lend itself to being used as a data source, which means more development is required to change it into an appropriate form. And actually designing the database is a valuable skill in its own right, which can take a considerable amount of time to develop.

Database round-trip: When a user requests a static page from a Web server, that Web server immediately sends the page back to the client. When a user requests a dynamic page that requires data from a database, the Web server must first make a request to (or *query*) the database for the necessary data, and then wait for it to arrive before it can assemble and send the page the user requested. This extra round-trip means a slight reduction in performance levels from the Web server. This delay might be unnoticeable on small Web sites, but may become more obvious on enterprise Web sites where thousands of pages might be requested per minute.

■ **Tip** Although you can't ever completely compensate for the additional round-trips that are made between the Web server and database, you can try to minimize the number of trips made by caching pages when they're created. After all, if the data in a page hasn't changed, why does it need to be retrieved from the database again? You'll look at improving the performance of a data-driven Web site with caching and other techniques in Chapter 12.

Dependence on the database: Using a database in a Web site means that should the database fail for some reason, the whole Web site will fail. The solution may be to run failover servers with synchronized databases, but as you can see from the next point, that could put quite a large dent in your pocketbook.

Cost: Full enterprise-level database solutions don't come cheap. At the top end of the market, Oracle Enterprise Edition starts at $40,000 and SQL Server Enterprise Edition at $25,000 for installation on *one* computer. Obviously, not everything costs that much, and indeed the databases used in this book are free, but things can get quite pricey quickly.

■ **Note** Even at a grassroots level, Internet service providers (ISPs) will offer some sort of database use in their hosting packages—typically MySQL or SQL Server—but charge an additional fee per month. Don't forget to check exactly how much, even if you aren't planning to deploy a data-driven Web site immediately. Having the facility in place is always a plus, even if it costs a little more. Of course, hosting your own Web site would solve that problem, but then that costs money to set up as well.

All in all, the decision comes down to how big your Web site is likely to be and whether you would be happy tweaking HTML all day for the rest of your life, rather than putting the effort in initially, letting the database do most of the tweaking for you, and generally enjoying the social scene. You're still a database fan, aren't you? I thought so.

How Do Web Sites Use Data Sources?

So then, you have a database or some other data source, and you have an ASP.NET page. What does the page do to use the data source?

As you know, a static page doesn't do a great deal. It's static. It has a pretty simple structure: a `<head>`, a `<body>`, and probably some headings and paragraphs. When you add ASP.NET, that page becomes dynamic and can be generated in many ways, depending on the code you add. However, it still ends up with the same basic HTML structure displayed by the Web browser. The difference is generally the content.

Database Uses in a Web Environment

At its simplest, using a data source in a Web site means getting it to store and then provide the content for a page. You define a page whose structure and layout don't change but whose content does. It becomes a template for you to fill in the gaps with information from the database. For example, Amazon.co.uk (`http://www.amazon.co.uk`) uses databases to store all the information and feedback for every product it sells, and yet, as Figure 1-1 demonstrates, the basic product page has the same layout, regardless of the item you're viewing.

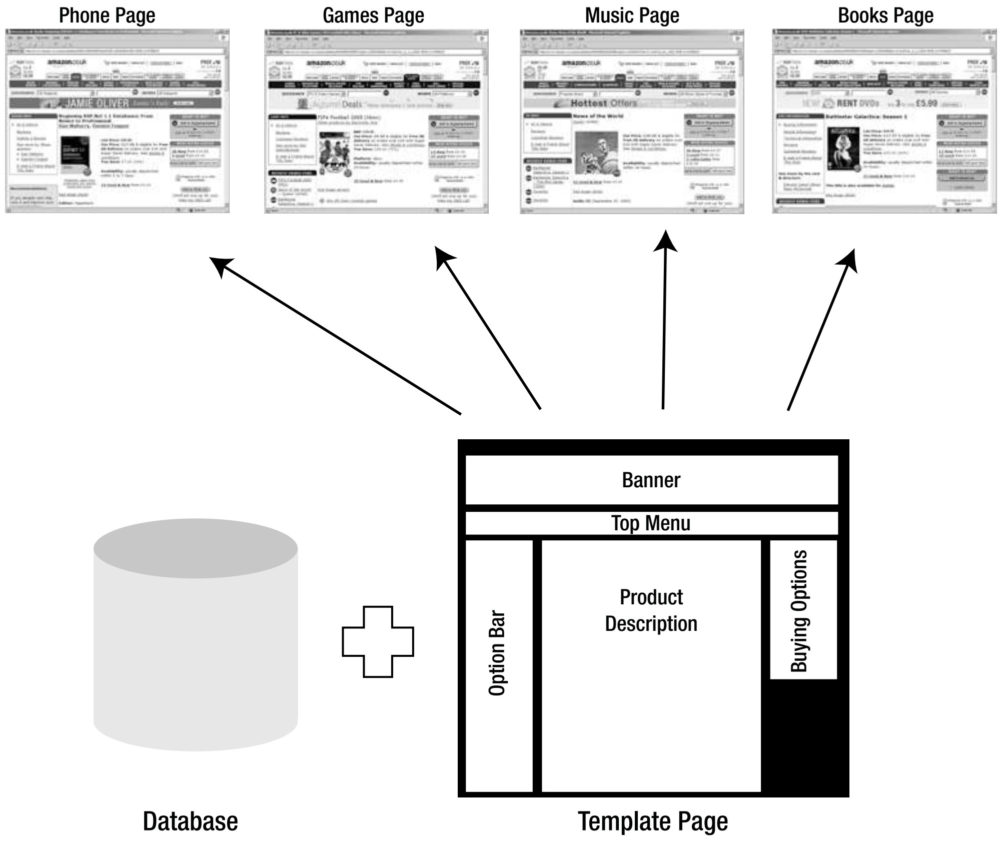


Figure 1-1. *Amazon.co.uk uses one template page and many product pages.*

Amazon.co.uk is also a good example of using a database to store layout and preferences information that changes depending on the page being requested. For example, the basic banner changes color according to the type of product you're browsing, and a combination of cookies and database data stores information about the items you've browsed and bought in the past, so Amazon.co.uk can suggest other content you may like in its recommendation pages.

In portal Web sites such as Slashdot (`http://www.slashdot.org`), a cookie on your machine identifies who you are to the Web site, and user preferences stored in the Web site's database allow for more radical changes to the Web site's user interface (UI), keeping track of which article groups you're interested in and whether the UI should be text only or full graphics, as Figure 1-2 demonstrates.

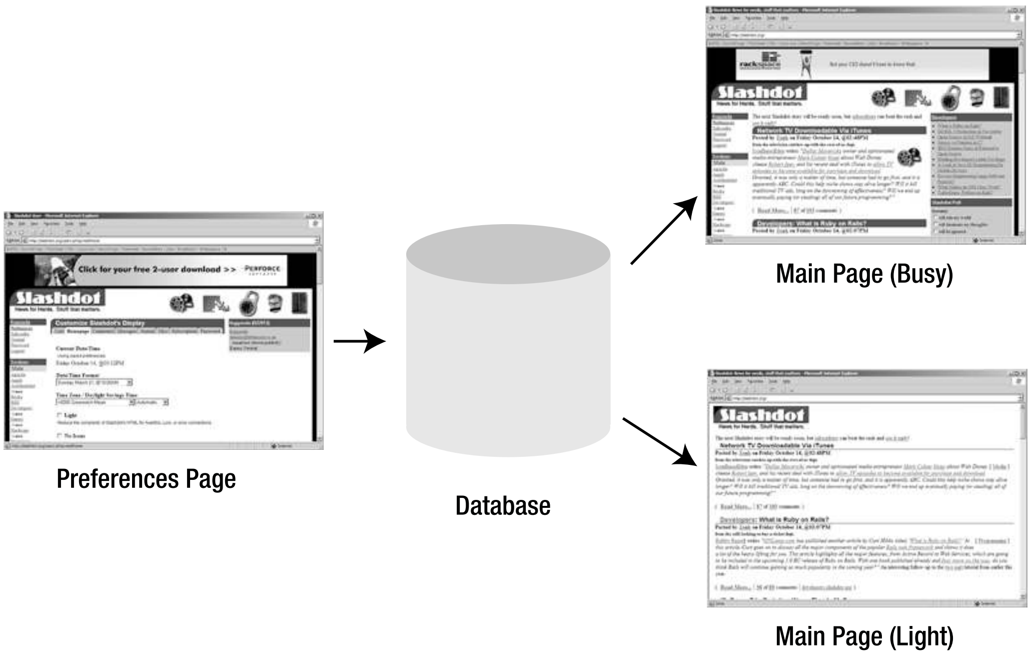


Figure 1-2. *Slashdot uses a database to display its pages according to your preferences.*

Delivering content and keeping track of user preferences aren't the only uses for a database in the Web environment, but they give you the idea. What about login systems, shopping carts, search engines, and bug-tracking systems? They're all variations on a theme, implemented as a database with a Web front end.

How Does the Web Site Get the Data?

It's time to geta little more technical. What actually happens when a data-driven page is requested by a browser? Does the code need to pray to the database gods for enlightenment and a source of knowledge? Of course not. Aside from anything else that ASP.NET may be doing in a page, the task of communicating with a data source takes just three steps, as shown in Figure 1-3.

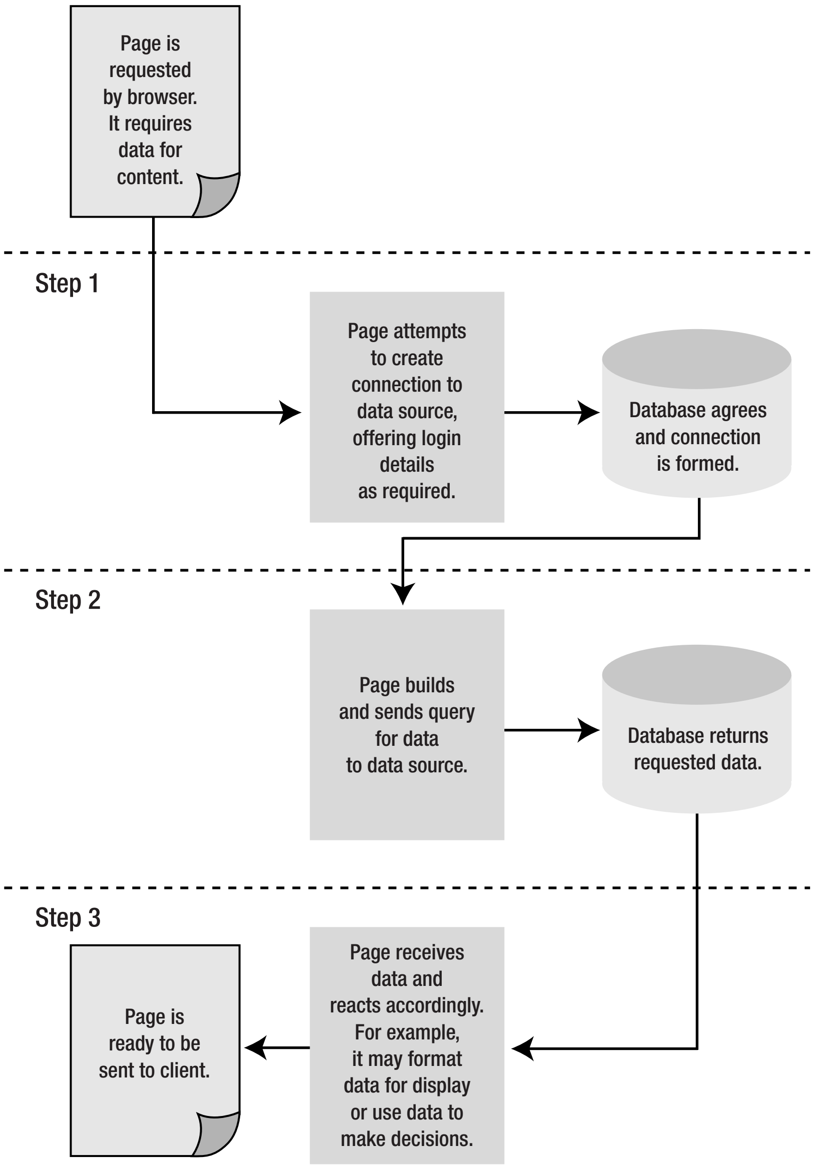

Page is requested by browser. It requires data for content.
Step 1
Page attempts to create connection to data source, offering login details as required.
Database agrees and connection is formed.
Step 2
Page builds and sends query for data to data source.
Database returns requested data.
Step 3
Page is ready to be sent to client.
Page receives data and reacts accordingly. For example, it may format data for display or use data to make decisions.

Figure 1-3. *It takes three simple steps to retrieve data and use it in a page.*

The steps are as follows:

1. The page tries to open a connection to a database. The code tells the page which database and where it can be found. In the examples in this book, the databases will be stored on the same machine as the Web server, but this doesn't have to be the case.

2. The page sends a query (also known as a command or statement) to the database. Usually, it's a request for some data, but it could be to update some data, to add some new data, or even to delete some data.

3. The database sends back some information that the page must then handle accordingly. If it's information for display on the page, it's rendered on the page, as with the earlier Amazon.co.uk and Slashdot examples. On the other hand, it may be a confirmation from the database that it updated, inserted, or deleted some data as you requested, or something else that doesn't directly affect the display of the page. It depends on the query sent in step 2.

These three steps are relatively simple, and once you have mastered them, you'll understand the basic requirements for building data-driven Web sites.

Now that you know what a page does when it talks to a data source, it's time to look at what you can actually use as a data source to drive your pages.

Looking for Information

Oranges aren't the only fruit. Databases aren't the only source of data you can use in your Web sites, and it's good to remember this as you start to develop your code. After all, why confine yourself to one central source if it doesn't make sense to do so? For example, you could put your lists of things to do today in a database, but it makes more sense to keep these items as a group of Outlook tasks, as a note in OneNote, or even as a simple text file. With a little tweaking, you can also use these as data sources for your pages.

The following sections cover the five general types of data sources you can use with your Web sites. First, let's look at the data source that you'll be using in this book: database servers.

Database Servers

Hang on, they're just called *databases*, aren't they? Where do the *servers* come into it? Well, strictly speaking, the database software you install—such as SQL Server, Oracle, MySQL, and so on—is a database server, or a database management server, depending on the product you're using. The database itself is just the collection of data you're filling your Web site with, and the server software is what hosts and manages it for you. A server may host many databases at a time, or just one, depending on the server configuration and the size of the database. The database that you'll use in the examples in this book is a few hundred kilobytes (10^5 bytes). One of the largest databases in the world is that of the U.S. Army logistics division, which is almost a couple of hundred terabytes (10^{14} bytes)!

The following are several different types of database servers (and therefore databases), and each works in its own way:

Relational databases: A relational database is the most common type and the one you'll use for the duration of this book. Information about an object or an event is strongly typed (just as .NET variables are) and stored in tables. Each table is given a set of rules as to how it relates to other tables in the database. For example, if you wanted to store some contact information in a database, you could create a table called ContactInfo and include items in it such as your contacts' first names, last names, e-mail addresses, work phone numbers, and Web site URLs. You'll spend Chapter 2 looking at what makes up a relational database, how you store data in it, and how you define relations and rules on that data.

Object-oriented databases: Information about objects and events are stored as objects and manipulated using methods attached to that object's class. This mimics the way you work with objects in .NET.

Object-relational databases: Almost a superset of relational databases, object-relational databases store information in tables, but the tables themselves are given a type and allowed to be operated on in a pseudo-object-oriented fashion.

Native XML databases: These store information about an object or event as an XML document rather than an object or row in a table, reading in and offering information as XML documents only. This kind of database has come into being only recently, following the development of XML as a popular technology.

A lot of commercial database servers are available. More relational and object-oriented databases exist than the others, but XML is gaining a great deal of popularity, so the balance is swiftly being redressed. However, the underlying query technology for XML has yet to be developed fully, so it will be a while before native XML databases are truly as powerful as relational and object-oriented databases.

Flat Files

The information stored in a database is often interrelated. For example, information about cats may be related to information about their owners. You design it that way, and database servers ensure that the relationship is maintained. On the other hand, the information in a flat file doesn't have a "flat file server" to keep track of a relationship between data in two flat files. The information about cats in one flat file may be related to the information about people who own cats in another flat file, but there is nothing to enforce that relationship. If an owner moved, a database server would note that the cat's address also changed. In the flat file, the cat would still be living at the old address.

A flat file lives in its own world and is unaware of any events related to the information it contains. That doesn't mean you can't use it as a source of data, though. Information is usually stored in lists or as comma-separated values (CSVs) such as the following, with each line storing data for one item:

```
Judy, tabby, 12, kitekat
Fred, ginger tom, 2, cat chow
Gene, siamese, 5, live mice
Ann, albino, 8, dog food
```

For example, you can use any of the following types of files in a Web site:

- Text files containing information written in a uniform way—perhaps as a CSV file or as a list of items (such as phone numbers), each on its own line
- Spreadsheet files generated by applications such as Excel and Lotus 1-2-3
- XML files

Web Services

You can also retrieve information from other systems that aren't directly connected to the Web server hosting your Web site. *Web services* allow you to expose functionality on remote servers and call that functionality across the Web. They're also cross-platform, allowing you to call functionality that is running on other operating systems (such as Linux) and other development platforms (such as Java).

You can make use of Web services by returning a data source from the remote server and using it as though it were a local data source. If the remote server is also using .NET, you can return a `DataSet` and use this as your data source. If the remote server isn't running .NET (perhaps it's a Linux machine), you can return an XML document and use that as the data source.

Objects

The three data sources that we've looked at so far all have one thing in common: they exist outside the code that uses them. The database is a stand-alone application, a flat file exists in the file system, and so on. It is also possible to use *objects* created within code as a data source, provided they're collections and implement the `IList` interface.

You can use objects for a whole host of different tasks, but a common use is as a wrapper around a set of database entities, where you'll use a collection to store a set of objects. A good example of this is in an e-commerce Web site.

In an e-commerce Web site, each product will be represented as a `Product` object, and a collection of products will be stored as a `ProductCollection` object. The `ProductCollection` implements the `IList` interface and can be used as a data source. If you want to display a list of products, you can pass the `ProductCollection` object to a `GridView` and use it as a data source.

Services

Your computer maintains a lot of information about itself, even if you never use it. It maintains user profiles, hardware profiles, e-mail archives, and more. They can all be used as data sources if you know where to look and how to access them. For example, you can use the following in your Web sites:

- You can tap into an Exchange server and search for messages, contacts, and calendar information.
- You can tap into the Windows registry, search for system settings, and tweak them if you like.
- You can tap into a network's Active Directory and work with users, groups, and other network resources.

Please be careful if you decide to start working with these kinds of data sources. A lot of security measures guard these services, and with good reason. Even with the best intentions, altering and deleting pieces of the registry, for example, can render Windows inert.

Introducing ADO.NET

So, you have a set of pages that need information and a data source to provide it. You know that the Web server will use the information to provide the page's content and influence the way the page displays. You need to tell the page how to retrieve the content from the data source and what to do with it afterward. Does this look like a job for ASP.NET?

Not quite. While it's true that you'll use ASP.NET to react and work with the information once it has been pulled from a data source, you actually use its sibling technology, ADO.NET, to work directly with the database. If you've worked with classic ASP, the relationship between ASP.NET and ADO.NET is the same as the relationship between ASP and ADO; the former deals only with the creation of pages, and the latter deals solely with retrieving information from data sources.

■**Note** While you may be thinking that using ADO.NET will be the same as using ADO, it's not just the "same old thing." Even though ADO.NET shares the same name as ADO, it isn't a simple evolution of ADO. You've already discovered that moving from ASP to ASP.NET was a complete paradigm shift in terms of how you build Web sites, and you'll soon see that moving from ADO to ADO.NET is a similarly large paradigm shift. A lot of the terminology is the same—you still have connections and commands, for instance—but that's about where the similarities end.

Data Access Technology: A Brief History

ADO.NET is the latest in a long line of Microsoft data-access technologies spanning a good ten years with the same aim in mind: to make database access as easy and as painless as possible for anyone who needs that facility.

Back in the late 1980s and beginning of the 1990s, database server vendors all faced the same problem. A lot of third-party vendors wanted to build products backed by a database but didn't want to be limited to using just one database. They wanted to keep the product as generic as possible so customers could use their application backed by their database of choice. The problem was that every database had its own way to access the data inside it, so a third-party vendor had to write new code each time it wanted to support a new database.

The solution the database vendors came up with was called Open Database Connectivity (ODBC). This is a common set of functions and interfaces agreed upon by all the major database vendors at that time to be implemented by all their servers. Third-party vendors needed to write code only against ODBC methods to access a database, and it would then work against any database that supported ODBC, which they all did. The third-party vendors were happy because the size of their products was reduced quite dramatically, and they all worked against every ODBC database. The database vendors were happy because third-party products worked against their database servers, and they could charge the third parties license fees. Everyone's customers were happy because of the increase in competition, product, and, well, wasn't it nice when everyone played nicely with everyone else?

ODBC was extremely successful and is still supported by all the major database servers in use today. However, one thing you can't call it is simple to use. ODBC works at quite a low level. In context, if your program spoke English, you would need to train yourself to write code in the Swahili that ODBC spoke when you wanted to access a database. Microsoft saw this problem and attempted to fix it by creating OLE DB, a set of Component Object Model (COM) components designed for Windows developers. OLE DB makes accessing data a bit simpler—more Spanish than Swahili, so still not English, but easier to learn. It also doesn't presuppose that the data source is a database, as ODBC does. OLE DB was pretty successful and is still supported by several vendors including Microsoft, which decided that OLE DB would be the cornerstone for its Universal Data Access (UDA) strategy.

One of the aims of UDA was to bring an object-oriented interface to ODBC and OLE DB, which were procedural in nature—more C than C++ or Visual Basic (VB). Its first attempts—Data Access Objects (DAO) and Remote Data Objects (RDO)—were designed to work against Access and larger databases such as SQL Server and Oracle, respectively. However, ADO version 2.0 superseded both of these in 1998. ADO is a technology originally designed to give classic ASP pages a way to access databases.

ADO.NET now takes over from ADO. Like its predecessor, ADO.NET gives you the ability to work with a data source through a common set of methods and interfaces, regardless of whether it supports ODBC, OLE DB, or its own proprietary access solution. This is achieved through a set of data providers, which are described in the next section. ADO.NET also provides better support for the following:

- Working with data away from the database itself or, rather, pulling information onto the Web server and working with it there instead of on the database server. This method of using *disconnected data* can improve performance if used wisely.
- The database when it is under attack from a large number of simultaneous queries. Stability and performance have been significantly improved in ADO.NET compared to ADO.
- Binding information to any control on the page, as you'll see in Chapter 6. Strictly speaking, this is more an ASP.NET feature than an ADO.NET feature, but it's an important capability when you're creating data-driven Web sites.
- With the introduction of the data source controls in ASP.NET 2.0 (discussed a little later in this chapter), it has become even easier to write data-aware pages than ever before.

In short, ADO.NET is a lot better than ADO ever was, and it's part of the .NET Framework, which provides a number of development benefits. Developing data-driven Web sites has never been so straightforward. The first release of the ASP.NET made writing data-aware pages almost idiot-proof, and with ASP.NET 2.0, it has become almost child's play. I say "almost" because data access is the one area where you're guaranteed to make mistakes when developing your Web site. Although ASP.NET 2.0 makes developing data-aware Web sites a lot safer, it's still possible to make mistakes.

Data Providers

You know now from your brief history lesson that the hard part of retrieving data from a data source has always been trying to talk to the source. That's why ODBC, OLE DB, ADO.NET, and the rest were created in the first place. Ironically, as you saw earlier, all you really need to do with the data source for a Web site can be reduced into three steps: creating a connection, sending a query, and dealing with the result of the query. ADO.NET provides a common interface for performing those three steps, regardless of whether you're using ODBC, OLE DB, or some other method to access it. It does this with *data providers*.

In this book, you'll see how to use three different data sources, so you have a choice when it comes to working on your own Web sites. MySQL has an ODBC interface. A Microsoft Access database (MDB) file has an OLE DB interface. SQL Server has both ODBC and OLE DB interfaces, as well as its own optimized set of access methods.

Suppose you're going to use MySQL. Writing a bit of pseudo-code for talking to this database, your three steps may look like this:

```
<%@ import Namespace="System.Data" %>
<%@ import Namespace="System.Data.Odbc" %>
<html>
  <script>
    create OdbcConnection object to link to MySQL
    create OdbcCommand object to set up and send a query to MySQL
    get returned an OdbcDataReader object containing the results of the query
    deal with the results....
  </script>
...
</html>
```

If you're using an MDB database file, which has an OLE DB interface, your pseudo-code might look like this:

```
<%@ import Namespace="System.Data" %>
<%@ import Namespace="System.Data.OleDb" %>
<html>
  <script>
    create OleDbConnection object to link to MDB file
    create OleDbCommand object to set up and send a query to MDB file
    get returned an OleDbDataReader object containing the results of the query
    deal with the results....
  </script>
...
</html>
```

The two pieces of pseudo-code are almost identical. The only difference is that the names of the objects are slightly different to correspond to the interface being used. So, for example, it's `OdbcConnection` for the ODBC database and `OleDbConnection` for the OLE DB database. As you'll see in Chapter 4, if you look at the real code, the actual calls you make are identical, which means the only things that change between different interfaces are the namespace to use and the names of the objects being used. The same is true if you want to use the native access method for SQL Server. The same method calls and steps are taken, but slightly different namespaces and objects are used, as in this example:

```
<%@ import Namespace="System.Data" %>
<%@ import Namespace="System.Data.SqlClient" %>
<html>
  <script>
    create SqlConnection object to link to SQL Server
    create SqlCommand object to set up and send a query to SQL Server
    get returned an SqlDataReader object containing the results of the query
    deal with the results....
  </script>
...
</html>
```

What you're seeing is the common interface for data-access operations provided by ADO.NET. A common set of base classes and interfaces (one for a Connection object, one for a Command object, and so on) are implemented *and optimized* for each data-access method you may need to use: ODBC, OLE DB, native SQL Server, and so on. Each group of objects is called a *data provider* and is housed in its own namespace. For example, Microsoft ships all of the following data providers:

- `System.Data.Odbc` is the .NET data provider for ODBC-based databases.
- `System.Data.OleDb` is the .NET data provider for OLE DB-based databases, such as Microsoft Access. You can also use it to access flat files.
- `System.Data.OracleClient` is the .NET data provider for Oracle databases.
- `System.Data.SqlClient` is the .NET data provider for SQL Server.
- `System.Data.SqlServerCe` is the .NET Compact Framework data provider for SQL Server CE. As you may imagine, its use is limited to applications running on personal digital assistants (PDAs) hosting an instance of SQL Server CE.

And that's one of the beauties of ADO.NET. As long as you know which data provider to use, you need to learn only one set of calls, and every data provider supports that set. Figure 1-4 shows this diagrammatically.

Going one step further, if a vendor would rather have .NET developers use a data provider specifically designed for its database server, instead of the generic OLE DB or ODBC one, all the vendor needs to do is implement the same common set of objects that every other data provider includes. An extra method or property here and there may take advantage of a particular feature unique to the data-access technology being modeled by the data provider, but in general, all the objects have the same methods and properties.

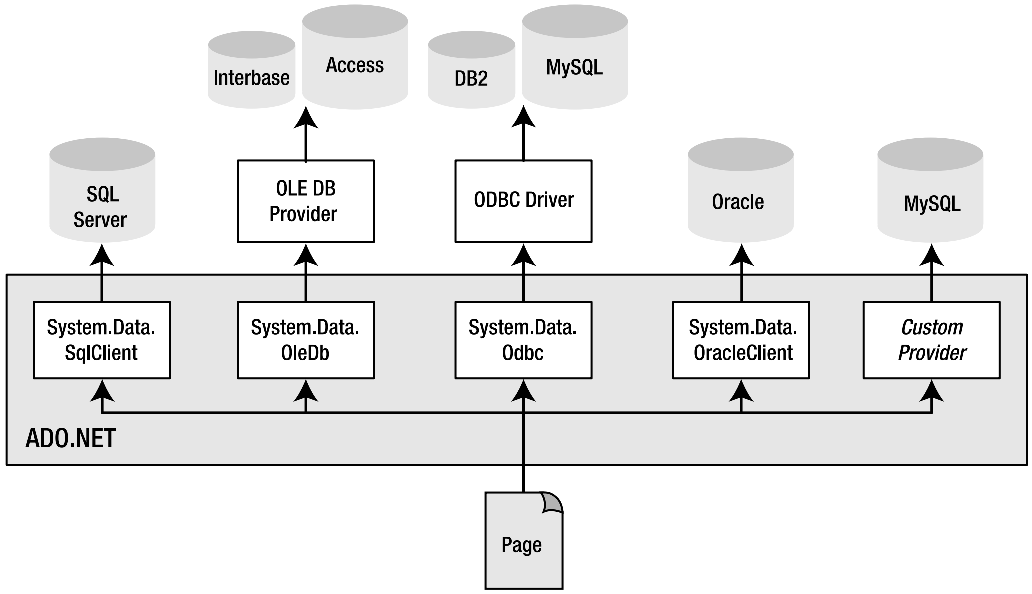


Figure 1-4. *Using data providers means you have easy, optimized access to all databases.*

Every data provider contains implementations of the following:

- **Connection object:** Used to represent the connection between the page and the data source.
- **Command object:** Used to represent the query to be sent to the database. Queries are much like functions and may use parameters filled at runtime rather than hard-coded values filled at compile time. To represent the parameters in a query, a data provider also includes a Parameter object.
- **DataReader object:** Used to represent the data returned by the data source as a forward-only, use-once result of a query. You'll start working with the DataReader object in Chapter 4.
- **DataAdapter object:** Used to populate a `DataSet` with the results of a query. Unlike with the DataReader, once the results are added to a `DataSet`, you can access it as many times as you like and however you want. Note that the `DataSet` isn't part of any data provider, and the same object is used, regardless of which data provider you use. A `DataSet` uses disconnected data on the Web server; it never works directly with a data source. The DataAdapter provides the bridge between the specific data provider and the `DataSet`. We'll look at the DataAdapter and `DataSet` objects in more detail in Chapter 5.

■ **Note** Although the objects that interact with the data source are the DataReader and DataAdapter objects, the DataAdapter tends to get short shrift when talking about data access. Although technically incorrect, it's common practice to refer to the DataReader and `DataSet` as the methods of accessing the database, and that's what I'll use for the rest of the book. Rather than saying DataAdapter/`DataSet`, I'll stick to just `DataSet`. Just be aware that wherever you use a `DataSet`, there's going to be a specific DataAdapter interacting with the data source.

- **Exception object:** Used to allow your page to fail gracefully if something untoward happens and lets you know (in detail) exactly what went wrong.

Figure 1-5 demonstrates how these objects fit into the grand three-step data access scheme you saw in Figure 1-3.

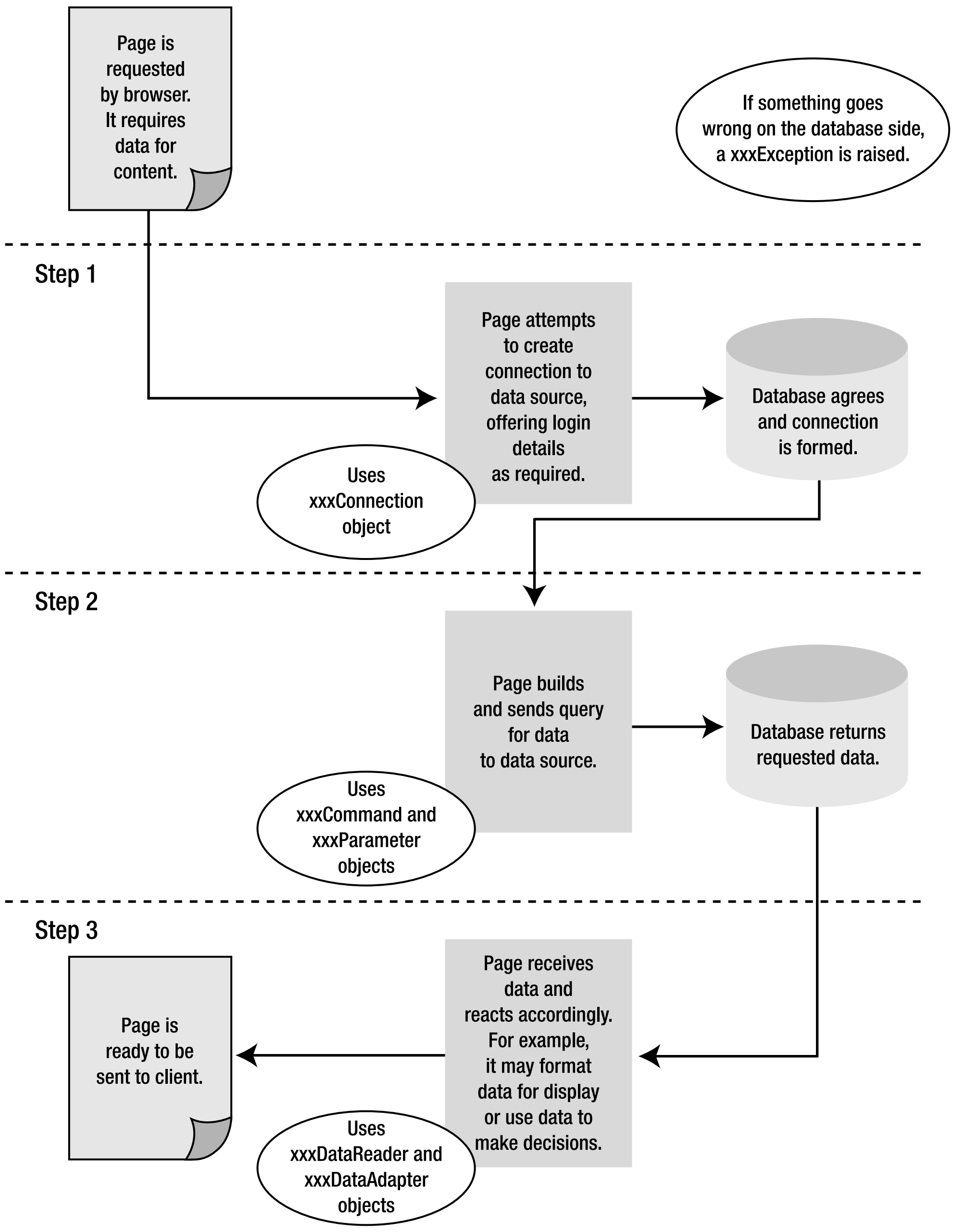

Page is requested by browser. It requires data for content.
If something goes wrong on the database side, a xxxException is raised.
Step 1
Page attempts to create connection to data source, offering login details as required.
Database agrees and connection is formed.
Uses xxxConnection object
Step 2
Page builds and sends query for data to data source.
Database returns requested data.
Uses xxxCommand and xxxParameter objects
Step 3
Page is ready to be sent to client.
Page receives data and reacts accordingly. For example, it may format data for display or use data to make decisions.
Uses xxxDataReader and xxxDataAdapter objects

Figure 1-5. *Putting the pieces of a data provider in their place*

Data Source Controls

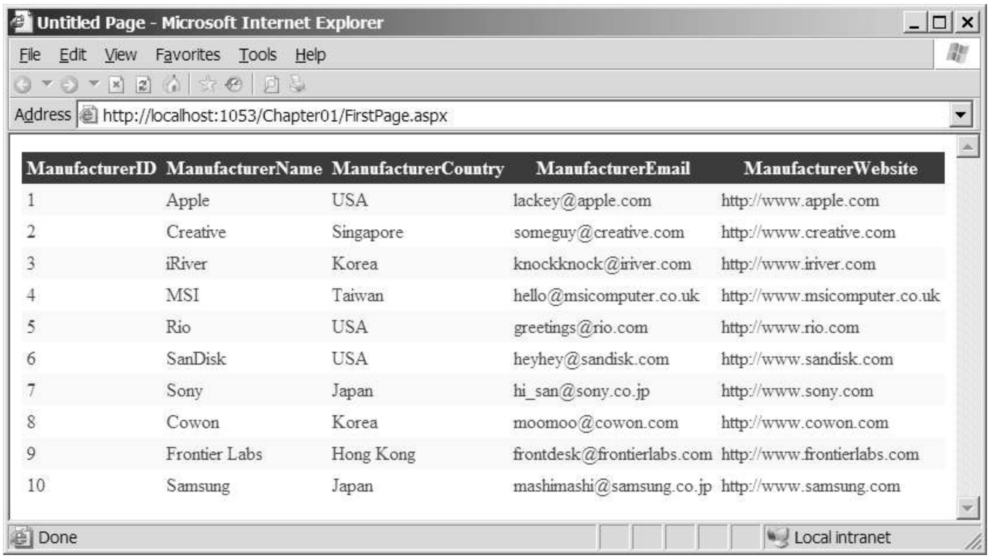
The paradigm and objects that we've looked at in the previous section were the only way to handle data access in the previous versions of ASP.NET. ASP.NET 2.0 introduces several new controls that simplify the way that you work with data.

The data source controls are a series of controls that allow the interactions with data sources to be handled without writing a single line of code, what I like to call *code-free data access*. Instead of having to create Connection and Command objects in code, you can create an instance of the `SqlDataSource` and point this at the correct data source. Any data source can be accessed, as long as there's a data provider available for it. The `SqlDataSource` handles the interaction with the data source for you automatically. In the next section, you'll see how these controls work.

Although the data source controls do provide an ideal means of building data-aware pages that allow you to view and modify data, you'll soon realize that there are limitations to what you can accomplish using this method. This is where the Connection and Command objects that you'll meet in Chapter 4 come into their element.

Developing Your First Example

Enough theory—it's time for your first example. You're going to discover exactly how easy it is to create a data-driven page by creating one in five seconds flat—well, slightly longer, as you have a little typing to do! The page that we're going to build will look very much like Figure 1-6.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:1053/Chapter01/FirstPage.aspx
Done
Local intranet

ManufacturerID	ManufacturerName	ManufacturerCountry	ManufacturerEmail	ManufacturerWebsite
1	Apple	USA	lackey@apple.com	http://www.apple.com
2	Creative	Singapore	someguy@creative.com	http://www.creative.com
3	iRiver	Korea	knockknock@iriver.com	http://www.iriver.com
4	MSI	Taiwan	hello@msicomputer.co.uk	http://www.msicomputer.co.uk
5	Rio	USA	greetings@rio.com	http://www.rio.com
6	SanDisk	USA	heyhey@sandisk.com	http://www.sandisk.com
7	Sony	Japan	hi_san@sony.co.jp	http://www.sony.com
8	Cowon	Korea	moomoo@cowon.com	http://www.cowon.com
9	Frontier Labs	Hong Kong	frontdesk@frontierlabs.com	http://www.frontierlabs.com
10	Samsung	Japan	mashimashi@samsung.co.jp	http://www.samsung.com

Figure 1-6. *Your first data-driven page in action*

For the purposes of this book, we're going to use free, or as cheap as possible, tools to build all of the examples. Both of the database servers that we'll use—SQL Server 2005 Express and MySQL 5.0, can be downloaded for free, and we'll use Visual Web Developer 2005 Express as our development environment.

Visual Web Developer 2005 Express is a cut-down version of Visual Studio 2005. Although it doesn't provide any of the high-end features of Visual Studio 2005, it does offer the same environment for building Web sites. Visual Web Developer 2005 Express provides code highlighting, error checking, and most of the features that you would expect from a full-fledged development environment. It also includes its own mini Web server (the ASP.NET Development Server), so those of you without access to a server running Internet Information Services (IIS) can work through the exercises in this book. At the moment, you can download Visual Web Developer 2005 Express for free from `http://msdn.microsoft.com/vstudio/express/vwd` for the "next year," and after that, you'll have to pay for it.

Although we're using the cut-down versions of SQL Server 2005 and Visual Studio 2005, if you happen to have access to the full-fledged versions, you'll also be able to work through all of the examples in the book. The differences are very minor—for example, some menus may be in slightly different places—so you should find it easy to follow along.

■ **Caution** If you're going to run this example (and the rest of the examples in this book), you'll need to install, as a minimum, .NET Framework 2.0 and Visual Web Developer 2005 Express. Refer to Appendix A for full instructions and come back here when you're ready to continue.

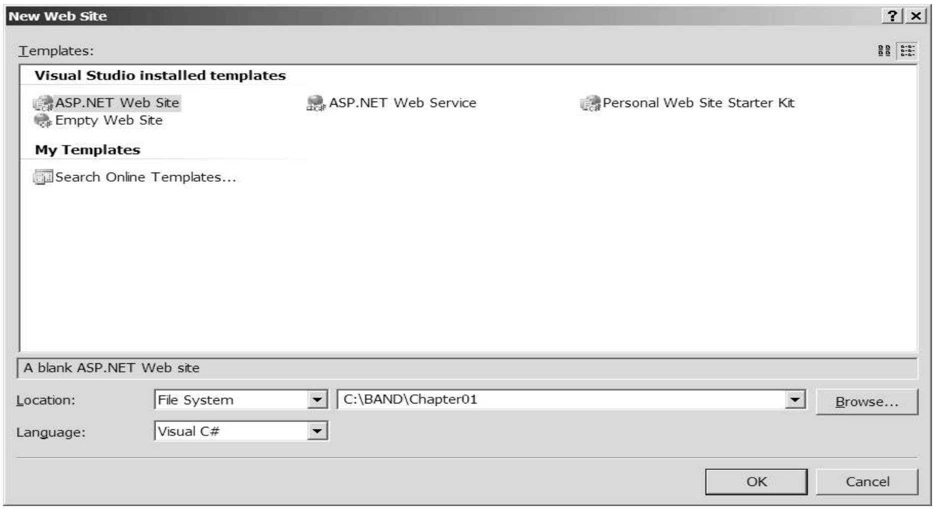
Try It Out: Creating a Simple Data-Driven Page

In this example, you'll create a simple data-driven page that pulls some information from a data source and displays it in a table on the page. In this case, the data source is a small MDB database file that you can find in the code download for this book (available from the Downloads area of `http://www.apress.com`). It's called `players.mdb` and should be copied to `C:\BAND`.

For this first example, we're going to use an MDB file as the database. In Chapter 2, we'll look at creating a database in both SQL Server and MySQL. When building real-world Web sites, you're advised to use a full-fledged database server such as SQL Server or MySQL, and avoid using MDB files.

To create the data-driven page, follow these steps:

1. Start Visual Web Developer and create a new Web site by selecting the File ➤ New Web Site menu option. This will launch the New Web Site dialog box, as shown in Figure 1-7.



A blank ASP.NET Web site

Location: File System C:\BAND\Chapter01

Language: Visual C#

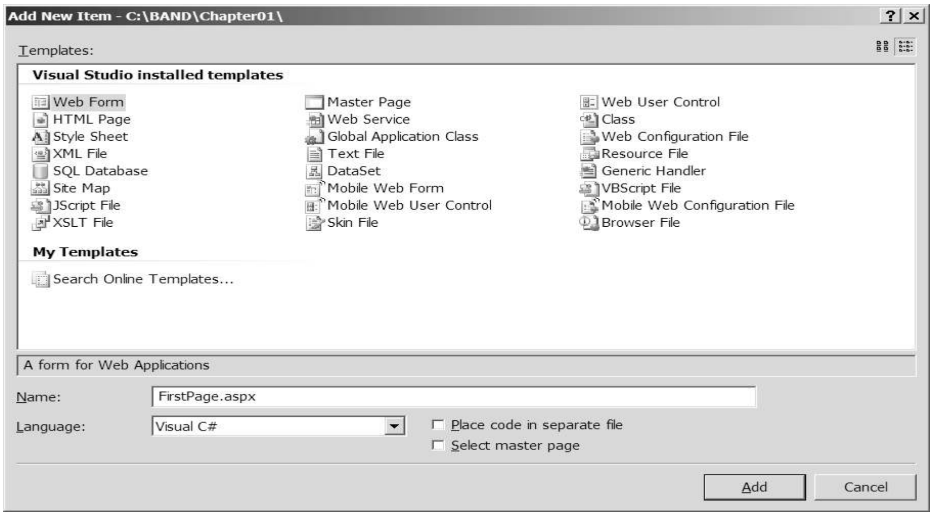
Figure 1-7. *The New Web Site dialog box in Visual Web Developer*

2. We're going to create an empty Web site for each chapter of the book, so select ASP.NET Web Site from the Templates list and enter `C:\BAND\Chapter01` for the location of the Web site. Because we're using the ASP.NET Development Server, our access to the Web site is file-based, so ensure that the Location option is set to File System. Make sure that the Language option is set to Visual C#. Then click OK to create the Web site.

3. By default, when creating a new Web site, Visual Web Developer will create a Web Form called `Default.aspx` in the Web site. We're not going to use this page, so right-click it in the Solution Explorer and select Delete from the context menu. Click OK in the confirmation dialog box to delete the page.

4. Right-click the folder name in the Solution Explorer and select Add New Item from the context menu. This launches the Add New Item dialog box, which allows you to create a whole host of different items, as shown in Figure 1-8.

5. Select Web Form from the Templates list. In the Name text box, enter `FirstPage.aspx`. Make sure that the Place Code in Separate File check box is *not* checked, and ensure that the language is set as Visual C#. Click Add to add the Web Form to the Web site. This will add the page to the Solution Explorer, as shown in Figure 1-9.



A form for Web Applications

Name: FirstPage.aspx

Language: Visual C#

☐ Place code in separate file

☐ Select master page

Figure 1-8. *The Add New Item dialog box*

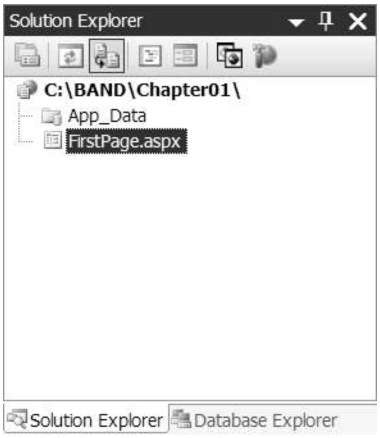


Figure 1-9. *Your new page (Web Form) has been added to the solution.*

6. Right-click `FirstPage.aspx` and select Set As Start Page from the context menu.
7. If `FirstPage.aspx` is not already open in the main window, double-click it in the Solution Explorer to open it.
8. Switch to Design view by clicking the Design tab at the bottom of the main window.
9. If the Toolbox does not appear on the left side of your screen, select View ➤ Toolbox to display it. Expand the Data entry in the Toolbox to see the data controls, as shown in Figure 1-10.

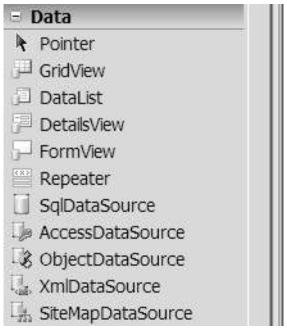


Figure 1-10. *The Visual Web Developer Toolbox contains a plethora of useful controls.*

10. Add a `SqlDataSource` control to the page.This will create an instance of the control called `SqlDataSource1` and open the Tasks menu for the control, as shown in Figure 1-11. If the Tasks menu is not shown, click the control to select it, and then click the little right-pointing arrow in the top right of the control to launch the Tasks menu.

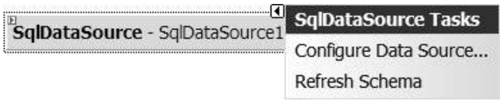
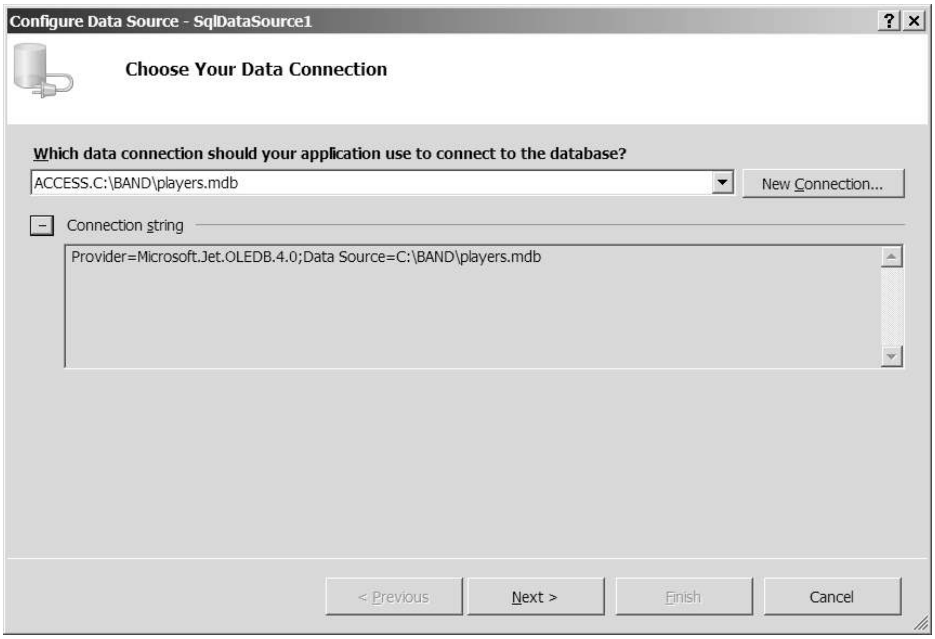


Figure 1-11. *Controls have Tasks menus showing common operations.*

11. Click the Configure Data Source option on the Tasks menu. This will launch the Configure Data Source wizard and allow you to select the data source to use.

12. On the first step of the wizard, click the New Connection button. We're not connecting to a SQL Server database, so select Change in the Add Connection dialog box. Select Microsoft Access Database File as the data source, and then click OK.

13. The database that we want to use is included in the book's code download, and you need to enter this as the database file name, as well as the correct location of the database. Specify the data connection to use as `C:\BAND\players.mdb`, and then click OK. This will create the correct connection string, as shown in Figure 1-12. Click Next to continue.


Configure Data Source - SqlDataSource1
Choose Your Data Connection
Which data connection should your application use to connect to the database?
New Connection...
Connection string
< Previous
Next >
Finish
Cancel

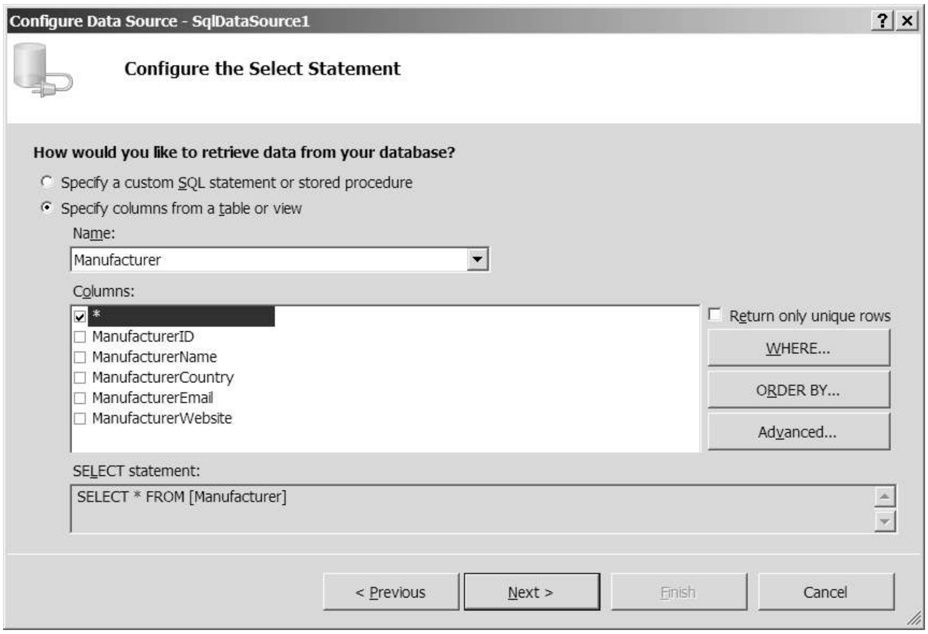
ACCESS.C:\BAND\players.mdb

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\BAND\players.mdb
```

Figure 1-12. *The data source has the correct connection string.*

14. On the Save the Connection String step, make sure the Yes check box isn't checked, and then click Next.

15. On the Configure the Select Statement step, select Manufacturer from the drop-down list and click the * column in the Columns list, as shown in Figure 1-13. Then click Next to move to the next step of the wizard.



Configure Data Source - SqlDataSource1

Configure the Select Statement

How would you like to retrieve data from your database?

- ○ Specify a custom SQL statement or stored procedure
- [✓ ● Specify columns from a table or view]

Name: Manufacturer

Columns:

- [✓] *
- [ ] ManufacturerID
- [ ] ManufacturerName
- [ ] ManufacturerCountry
- [ ] ManufacturerEmail
- [ ] ManufacturerWebsite

[ ] Return only unique rows

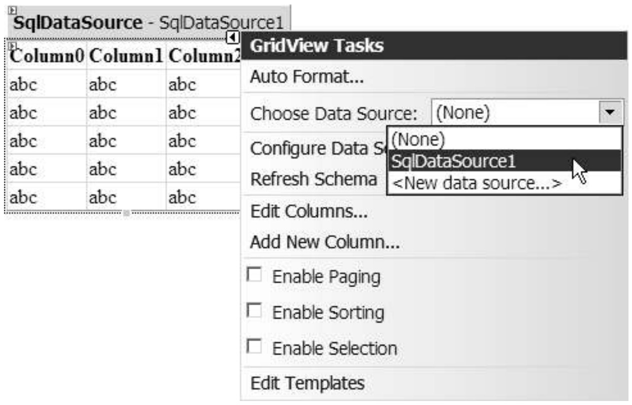
SELECT statement:

```
SELECT * FROM [Manufacturer]
```

Figure 1-13. *Specifiying the data that we require.*

16. If you wish, you can select Test Query to preview the data that the query will return. Click the Finish button to close the wizard.

17. Add a `GridView` control from the Toolbox to the page below the `SqlDataSource`. From the Tasks menu, select `SqlDataSource1` as the data source for the `GridView` from the Choose Data Source list, as shown in Figure 1-14.



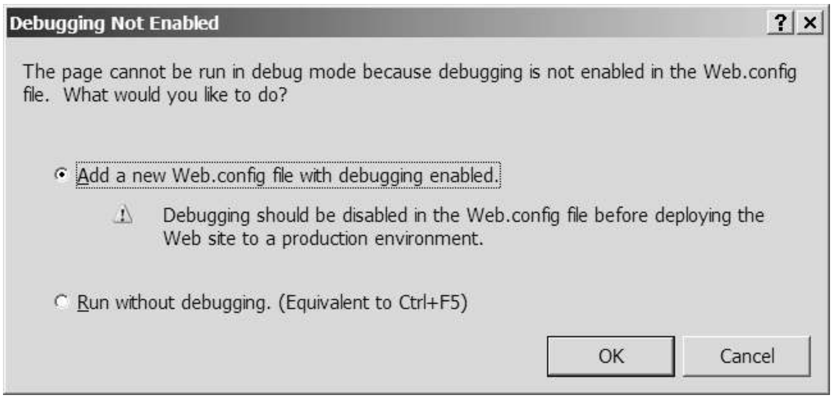
SqlDataSource - SqlDataSource1

Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

Figure 1-14. *Choosing a data source from the Tasks menu*

18. Reopen the Tasks menu for the `GridView` and select the Auto Format option. Select Colorful from the Auto Format dialog box, and then click the OK button.

19. Save `FirstPage.aspx`, and press F5 to debug the Web site. A dialog box will ask if you want to create a `Web.config` file to enable debugging, as shown in Figure 1-15. Click OK to create a `Web.config` file.



Debugging Not Enabled

The page cannot be run in debug mode because debugging is not enabled in the Web.config file. What would you like to do?

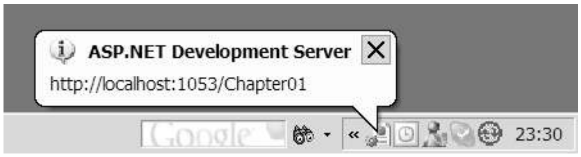
[✓ Add a new Web.config file with debugging enabled.]

Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.

Run without debugging. (Equivalent to Ctrl+F5)

Figure 1-15. *Enabling debugging in Visual Web Developer*

This will start the ASP.NET Development Server, as shown in Figure 1-16. Once the ASP.NET Development Server is running, all being well, you'll see a page in your Web browser looking none too dissimilar from Figure 1-6. Congratulations, you've just made your first data-driven page!



ASP.NET Development Server
http://localhost:1053/Chapter01

Figure 1-16. *ASP.NET Development Server is used by Visual Web Developer for debugging.*

How It Works

Visual Web Developer allows you to automate a lot of simple tasks when you use it. In this chapter's example, you built a Web Form (Microsoft's terminology for an ASP.NET page) that interacts with a database and displays the results in a graphical format. And you've done this without writing a single line of code!

You saw how to use Visual Web Developer to build a data-driven page using two of the new controls available with ASP.NET 2.0: the `SqlDataSource` and the `GridView`. You'll look at these and the various other data controls in more detail in the chapters to come.

It's time to finish this chapter before a bug crops up.

Summary

In this chapter, you've looked at the world of data-driven Web sites from 50,000 feet. You learned how a page interacts with a data source and the different types of data sources that interact with pages. Specifically, you learned the following:

- Databases generally support at least one of the ODBC and OLE DB common database interfaces, although some also support their own optimized data-access methods for speedier results.
- ADO.NET is a relatively new technology—part of the .NET Framework. It presents the developer with a set of data providers, each of which is optimized to access data through OLE DB, ODBC, and so on, but all of these data providers are used in the same way.
- The following are the three steps for accessing a database and using the information retrieved from it in a page:
  1. Create a connection to a database with a Connection object.
  2. Send a query to the database with a Command object.
  3. Handle the data appropriately depending on what you want it to do. Data is returned from the database using a DataReader or DataAdapter object and may be stored on the Web server in a DataSet object, which is generic and not specific to any data provider.

Finally, you created your first data-driven ASP.NET page using a simple MDB database file and Visual Web Developer. You saw how a Visual Web Developer wizard allows you to create a page in simple stages.

In the next chapter, you'll start coming down from 50,000 feet and investigate what makes a database, and in particular a relational database, tick.