AFTER YOUR QUERY IS DONE, the various query techniques discussed in the last chapters (such as ADO Command objects and the ODBC API) still need to pass back the results to the routines that initiated the query in the first place. More and more developers are choosing architectures that run queries at remote locations and transport the results back to another layer or tier. While this might sound like the traditional client-server rigs I’ve discussed before, the difference here is that the client is not opening a connection and managing a query using ADO. In many cases, the client is making a DCOM call to an out-of-process COM component running on a remote server. To do this, the remote COM component opens a connection, runs the query, and returns the data. In other situations, as in Web architectures, the client is simply a browser accessing an ASP page that uses ADO to run the query and return the data in the form of HTML, XML, and XSL or combinations of the above.

From the outset, ADO was designed to permit a middle-tier component to construct a Recordset, package it up for transport back to a client across process boundaries, and reconstitute the data stream as a Recordset on the client end. As you’ll discover in this chapter, returning a Recordset is not your only alternative—it’s not even your best alternative when returning the results of your query across process boundaries.

I’m of the opinion that less is more when it comes to passing data between layers. Because you wrote the query in the first place, and because you also defined the database and its tables (or at least understand the schema), you should also know how the data is to be returned. You don’t usually need to be told in any degree of detail how many columns were generated, the datatypes of each column, the updatability of individual columns, or which tables were used to retrieve the data. While this information is interesting to clients who don’t have an intimate knowledge of the data layer, it’s just overhead that describes schema information with which you’re already very familiar.

The other aspect of this DDL overkill is that the typical ADO Recordset repeats the same DDL information each time you run the query—not just the first time the query is run. It’s sorta like packaging individual potato chips in tiny plastic bags that have the size, weight, and composite ingredients printed on the outside. Wouldn’t it be great if we could requery and just get the data—and not all the extra
DDL? Perhaps someday, which is actually not too far away. The new .NET DataReader comes pretty close to this raw data form. You'll see how this new approach works in the last chapters of this book.

On the other hand, some developers ask for more metadata, to be able to use “spandex”\(^1\) properties, and to bind complex validation rules, etc. If you are going to compile and forget your application, less metadata is good for the network. However, for adaptable applications, DDL is crucial. Granted, the OLE DB layer has no thought of optimizing the transfer and caching of metadata. Perhaps someday.

For the most part, the new .NET philosophy is moving headlong into XML. That is, much of the data saved, persisted, and transported is in XML format. The schemas are represented and persisted in XSL right in your application. By no stretch of the imagination should you believe that this approach will yield better performance. It does lead to better application interaction, but XML data sets can easily take ten times more space to store or transport. Many developers seem to think that this approach is the answer to the universal question: "What is the meaning of life?" For those of you who don't accept "42" as the answer, you probably won't accept XML as the answer either.

That said, consider that XML is a very powerful new paradigm and here to stay. XML by itself will never replace ADO or COM, but the .NET Framework will. Remember when we old folks looked somewhat askance at HTML and ASP-based Visual Basic Script applications? Well, XML has the potential to be even more important. Stay tuned.

**Delivering Pizza with a Dump Truck**

Developers trying to move data from one place to another can be overwhelmed with the variety of choices. As ADO matures it adds even more choices as you saw in the last chapter. ADO.NET also adds a bevy of new options for data transport. Usually the “places” we expect to transport from or to include a component run on a server somewhere and a remote client application. As I see it, developers face a number of problems when trying to solve this problem.

The first is what delivery vehicle should be used. All too often, we choose an oversized delivery vehicle for the situation—like delivering pizza with a twelve-ton dump truck. It works, but it scares the customers, not to mention the neighbor's dog.

The second problem is that we tend to return far more data than the customer can consume before it starts getting cold. This is like delivering all the pizzas you anticipate the customer will order for the next two weeks at one time. At this point

---

1. By spandex, I mean the ability to add additional properties to the Field to further qualify, clarify, and extend the metadata.