1 ADO.NET and System.Xml Version 2.0

ONE OF THE FUNDAMENTAL design goals of ADO.NET and the classes from the `System.Xml` namespaces is to give the developer complete control over how data is accessed and used. In designing ADO.NET, Microsoft concentrated on providing the right core building blocks by factoring functionality into discrete, optimized objects, with the idea that classes could be built in later versions that use these building blocks to make common tasks easier.

When Microsoft first introduced version 1.0 of the .NET Framework, many developers familiar with ADO initially found this new factoring of classes for accessing and working with data unfamiliar. The one-size-fits-all `Recordset` object that formed the core of ADO and previous data access technologies was replaced with a series of new objects optimized for specific scenarios, such as the `DataReader` and `DataSet`. Meanwhile, a raft of new classes was added to the Framework for working with Extensible Markup Language (XML) documents and XML-formatted data. While this factoring provides added flexibility and control, many common tasks require understanding and tying together multiple independent objects, as opposed to the simplicity of the single `Recordset` object in ADO.

Now, in version 2.0 of the Framework, we get some exciting new additions to the ADO.NET and XML classes that build on the original version 1.0 classes, adding ease of use for common scenarios as well as introducing a host of new features. There is also support for the new features in the forthcoming version of SQL Server (code-named "Yukon") and the introduction of many core data access techniques that developers have been asking for since version 1.0.

This book covers all these new features. However, bear in mind that, as we described in the introduction to the book, the version of the Framework being released at the time of writing (late 2003) is a **Technology Preview** edition rather than a final release.

So, in this first chapter, we will:

- Overview the version 2.0 Technology Preview release
- Summarize the new data management features in version 2.0
- Look at the reasons for the growing importance of XML
- Summarize how the techniques for binding to data in ASP.NET are changing
- Provide a roadmap for the remainder of the book

We start with an overview of the Technology Preview release of version 2.0 of the .NET Framework. Bear in mind that this book is about what's new in version 2.0 of the .NET Framework, so we'll be assuming that you are familiar in general with the version 1.0 data access techniques and classes.

The Version 2.0 Technology Preview Release

The Technology Preview version contains the core changes planned for version 2.0 of the .NET Framework, including the provision of new features prompted both by Microsoft's own development team's inspirations and by feedback from the version 1.0 customer base. It also, of course, includes fixes to known issues with the version 1.0 classes and in some places a tidying up of the class interfaces.

But this is by no means a definitive view of what the final product will contain. It is a good general guide to the way the product is changing and to the kinds of features that will be encompassed in the release version (due toward the end of 2004).

So, the Technology Preview release—and this book—will help you as a developer to understand how data access in .NET is moving and to be more prepared for the changes when they arrive. It means that you can see how the Microsoft team is thinking about the future of the product and will also allow you to design your applications to suit this evolution. Just bear in mind the proviso that the final release version will no doubt be subtly different from what you see here.

Enhancements in Relational Data Management

Although some developers initially struggled with the move away from the simplicity of the single ADO `Recordset` object in version 1.0 of the Framework, the .NET relational data access model offers proven advantages in several key areas.

- As application development has moved from the connected client/server model toward Web-based applications, the focus has increasingly been on a disconnected data access approach. The .NET relational data access techniques provide a rich disconnected environment that makes it easy to remote data to clients, access data server-side, and perform staged or bulk data updates.
- The explicit factoring between objects that connect to and use database resources and objects that contain pure state provides the developer with more control over database interactions, performance characteristics, and application behavior.
- Programming in the .NET environment predominantly involves managed code, which runs under the control of the Framework runtime. The ADO.NET classes within the .NET Framework are exposed as fully managed code, providing a safe, consistent development platform with excellent error handling and debugging capabilities.
- The `DataSet` object introduced in version 1.0 provides a "container" for storing data from multiple tables, and optionally the relationships between these tables, together with associated classes and methods that provide easy updating of the source data and serialization to XML for transmission across networks. In version 2.0, new classes are added to the Framework to make this approach even more flexible.
- The use of data binding to display data from a variety of sources has grown in importance, especially when building Web applications. ASP.NET provides a range of controls that can perform data binding in different ways, and the `DataReader` object introduced in ADO.NET makes this a highly efficient approach for displaying relational data in grid or free-form formats.

- New classes provide a tighter integration with XML data and components.

These are only a few of the reasons that the .NET data access model is proving successful. There are plenty more new features in version 2.0 of the Framework. We'll be summarizing them later in this chapter, and then we'll go on to examine them in more depth in later chapters.

Enhancements in XML Data and Document Management

While the move toward Web-based application development has changed the requirements we have for relational data access, there have been more fundamental changes in the area of document and unstructured data management. Over the last few years, XML has become the de facto way to persist this kind of data in a way that is human-readable, can be machine-processed, and avoids custom binary representations.

Since its introduction, XML has evolved in several different directions. While the unstructured information storage paradigm is still extremely important, we've also come to see XML as a way to serialize and persist structured and hierarchical data (for example, rows and columns from a data table, or more than one related data table).

Prior to .NET, the usual way to manage XML was through the MSXML component, either via the XML Document Object Model (DOM) class, used to build an in-memory tree representation, or through the Simple API for XML (SAX) model, which "pushes" events to the application. Although this works well enough, .NET version 1.0 introduced several exciting new alternatives for manipulating XML data.

- A fast, forward-only, "pull" model XML parser named `XmlReader` that is able to read streams of XML in a highly efficient manner. This innovative API provides a simpler and more intuitive approach to reading XML documents. The complementary class `XmlWriter` is used to generate XML to a stream, including automatic indentation of the output and white-space control.
- An XML data store that is used to load XML documents into memory for editing and updating. This is the `XmlDocument` class, and it provides full standardization with the World Wide Web Consortium (W3C) recommendations for the XML DOM level 2.
- A cursor-style API for navigating through XML documents. The `XPathNavigator` class implements the XPath 1.0 specification for

querying XML documents. This is combined with a read-only store optimized for XPath queries, called the `XPathDocument`. In version 2.0 of the Framework, the `XPathDocument` is extended to provide more features, including change tracking and the ability to update data stores in a manner similar to the ADO.NET `DataSet`. The `XPathNavigator` is also extended to support the more powerful XML query languages for XPath 2.0 and XML Query Language (XQuery).

- The `XmlDataDocument` object, which acts as a bridge between XML and relational data by allowing data to be viewed and manipulated as XML or as a `DataSet`. While this class is useful, the new features of the `XPathDocument` will probably prove to be more flexible in this area, as well as allowing the data to be remoted across a network.
- A series of objects that allow developers to create XML schemas and validate XML documents against supplied XML schemas, along with objects such as the `XslTransform` class to perform transformations using XML Stylesheet Language Transformations (XSLT) stylesheets.

In version 2.0 of the Framework, there are many enhancements in the XML data management arena. In addition to improvements to the `XPathDocument` class, there are new features for working with XQuery and better integration with the new version of SQL Server ("Yukon") to be released around the same time as version 2.0 of the Framework.

Summary of New Features in Version 2.0

The remainder of this book looks in detail at what's new and the fundamental changes to existing techniques for working with relational and XML data in version 2.0 of the .NET Framework. We've divided the content into the two obvious sections:

1. The relational data management features in ADO.NET, covering the namespace `System.Data` and its subsidiary namespaces
2. The XML data management features in the .NET Framework, covering the namespace `System.Xml` and its subsidiary namespaces

These two sections provide a summary of what is new in both ADO.NET and `System.Xml`. However, bear in mind that one of the aims in the .NET Framework is to integrate data from independent data sources and data in different formats, so the distinction between the two approaches listed above is not absolute. Often you will work with XML data through the various ADO.NET objects or manipulate a relational data store using objects from the `System.Xml` namespaces.

Summary of New Features in ADO.NET

The new features of ADO.NET that are covered in this book are listed below.

- Multiple Active Results Sets (MARS) allows a connection to support multiple concurrent commands and multiple open sets of results (for example, multiple open `DataReader` instances). You can open a results set from a query, and then execute subsequent commands over the same connection while the previous one is still open and being accessed.
- The addition of truly asynchronous operations for connections and commands, allowing code to concurrently fetch and process data from multiple sources. This can provide a considerable performance boost in many situations—particularly when combined with the MARS feature.
- Support for paging data rows is now built in. The new `ExecutePageReader` method of the `Command` class provides more efficient access to data that is displayed in separate pages, rather than as a single list.
- New features for executing SQL batched commands. Commands that contain more than one SQL statement can now be executed in a more efficient manner with no extra work required from the developer.
- The provision of in-process server-side cursors for programmatic processing of data within a stored procedure. Because SQL Server "Yukon" can host the Common Language Runtime (CLR), you can write code that uses server-side cursors without losing out on performance to the same extent as traditional ADO methods.
- The addition of change notifications for SQL Server, allowing code to cache data and results sets, but automatically react to any changes that invalidate the original data within the database.
- SQL Server "Yukon" now supports User-Defined Types (UDTs), allowing you to build your own data types as .NET assemblies and install them into the database server and client so that information can be accessed in a more natural and efficient manner.
- A set of new classes, called ObjectSpaces, that allow data to be handled directly as objects, rather than as individual values.
- New classes in ADO.NET support programmatic batch updates and bulk data loading from various data sources into SQL Server.

Summary of New Features in System.Xml

The `System.Xml` namespaces have been extended considerably in version 2.0, in order to implement the main aims listed below.

- Integrating XML data with a range of data sources, and exposing a programming model that follows the same paradigm as that used in ADO.NET for reading and updating data. This is, in many ways, a parallel with the `DataAdapter` techniques used with the ADO.NET `DataSet` class but based on the XML data model rather than the relational data model.
- Adding built-in support for XQuery and XML Views. This provides a technique for distributed query processing over multiple different data sources, though for the current release this is targeted at SQL Server and XML documents. This is in effect the migration to .NET of the SQLXML 3.0 technology available in SQL Server 2000 today, but with several enhancements in the mapping technology and the query capabilities. Updates via XML Views are supported through the use of declarative "updategrams" that detail the changes that have occurred to the data.
- Providing better support for the storage of XML data and the integration of relational and XML data management techniques for the new version of SQL Server ("Yukon").
- Enhancing existing version 1.0 XML classes to support recent W3C standards to provide closer integration between the components in `System.Xml`; significantly improve the performance of XML parsing, XML schema validation, and XSLT transformations; and enable XML schema types for the classes.

To meet these aims, the `System.Xml` and subsidiary namespaces in version 2.0 of .NET contain several new classes and improvements to existing ones.

- There are major changes to the `XPathDocument` class. The content can now be edited, and change control is built in so that updates, inserts, and deletes are automatically tracked. It can also be remoted while preserving the XML data model and XML schema types.
- A new `XPathChangeNavigator` class is introduced to allow code to access the changes in an `XPathDocument`, and a new class named `XPathEditor` can be used to perform changes.
- The introduction of XML Views, a mapping between relational data types and XML data type definitions, allows relational data stored in SQL Server to be viewed and handled as XML.

- A new `XmlAdapter` class mirrors the concepts of the ADO.NET `DataAdapter` class. It connects an XML document to a database via an XML View, allowing data to be read and changes to be persisted to the database through auto-generation of SQL statements.
- Updates to the `XmlReader` and `XmlWriter` classes support the XML schema types, and a new `XmlFactory` class provides a mechanism for generating `XmlReader`, `XmlWriter`, and `XPathNavigator` instances based on user-defined settings.
- XQuery is supported through the new `XQueryProcessor` class, used in conjunction with an `XPathNavigator` class to query and return information from XML documents using the W3C XQuery language. This language has a more readable (non-XML) syntax for ease of authoring. XQuery is to XML what SQL is to relational databases. XQuery is set to become the preferred query language when querying over XML documents, XML Views, or any data source exposed as XML.
- The latest version of SQL Server ("Yukon") adds features that integrate with version 2.0 of `System.Xml`. This includes the provision of a new first-class data type for XML, the `xml` data type, allowing SQL Server to be used as a store for XML documents. This allows columns in a table to be marked as an `xml` data type, and XQuery queries can be performed over the XML and associated schema to provide type information.

These brief listings of the new features should whet your appetite to learn more, and that's what we'll be doing in the remainder of this book. To help you find what you need, a later section of this chapter presents a roadmap of the contents of the remaining chapters.

The Growing Importance of XML

You can see from the list of new data-oriented features in version 2.0 of the .NET Framework that there is considerable emphasis on XML as a data persistence, transmission, and manipulation format. XML was originally seen as a way to expose information that may or may not follow some structured or hierarchical format, and the XML standards allow XML to be used in a huge range of ways for storing information of almost any type.

As XML becomes more mainstream, it fulfills an increased role as an information integrator. XML is the bridge between relational data, objects, documents, Web Services, and more.

The Changing XML Landscape

The `System.Xml` version 1.0 APIs were developed at a time when XML standards were quite different than those that exist today. The predominant standards at the end of 1999 and early 2000, when the initial APIs for version 1.0 were laid down, were XML 1.0, DOM 1.0 and 2.0, XPath 1.0, and XSLT 1.0. Significant standards such as the **XML Information Set** (October 24, 2001, http://www.w3.org/TR/xml-infoset/), **XML Schema** (May 2001, http://www.w3.org/TR/xmlschema-1/), and **SOAP** 1.2 (CR December 2002, http://www.w3.org/TR/SOAP/) subsequently emerged. These standards are having a positive impact across the industry.

In the last two years there has been a significant shift from viewing XML purely as a text-based format to a data representation format epitomized by the XML Information Set specification, which was developed in recognition of the fact that the DOM API did not define the relationships between the information item types found within an XML document. An XML Information Set (often described just as an **XML InfoSet)** refers to the data types that can be found in an XML document after it has been parsed by an XML parser. This shift of emphasis toward XML defined in terms of information items now appears in many of the W3C specifications. For example, to quote from the SOAP 1.2 specification:

> A SOAP message is specified as an XML Information Set or XML InfoSet (http://www.w3.org/TR/soap12-part1/#XMLInfoSet#XMLInfoSet). While all SOAP message examples in this document are shown using XML 1.0 syntax, other representations MAY be used to transmit SOAP messages between nodes (see SOAP Protocol Binding Framework, http://www.w3.org/TR/soap12-part1/#transpbindframew#transpbindframew).

What's more, this has been reflected in all of the significant emerging W3C specifications, such as XQuery, XPath 2.0, and XML Schema 1.0. Each of these specifications is dependent on the definition of XML as InfoSet items.

To draw a parallel: When working with a relational database, the way the data is actually written to disk (serialized) is completely hidden from the developer and the database administrator. This is desirable because it means that the method in use can be changed over time to make storage more efficient without breaking existing code. Rather than having to know about tracks and sectors on the disk, the developer or database administrator deals with the database through a level of abstraction that consists of tables, rows, and columns. This is the relational data model.

The XML InfoSet performs the same role of abstraction. The data may or may not be written as serialized XML 1.0 with angle brackets, but either way the data types that are exposed always appear consistent to the processing application. Handling XML InfoSets provides the ability to expose a multitude of data sources as XML, a technique often referred to as **virtualization**. The `XPathNavigator` class provides XML virtualization over a data source in `System.Xml` today, based on the XPath 1.0 data model, which uses the XML InfoSet item types.

Perhaps one of the most significant specifications to emerge recently is the **XQuery and XPath 2.0 Data Model** (http://www.w3.org/TR/query-datamodel/), which, being based on the XML InfoSet, defines the data model for the XQuery language in the same fashion as the relational data model does for the SQL query language. In many ways this should really be considered *the* XML data model, since its importance goes beyond the bounds of just the XQuery and XPath 2.0 languages.

XML and Relational Databases

Relational databases such as SQL Server are the origin of much of the data that is interchanged today and then typically stored again in other relational databases. Since relational systems today are overwhelmingly used to manage structured data, most of the XML exchanged also fits the relational model of structured data. Hence it is becoming increasingly common for XML to be focused in scenarios that involve structured data storage and transmission, for example, exposing relational data (the traditional "rows and columns" format), and hierarchical data (such as multiple related data tables).

This movement has also tended to simplify the XML content by concentrating mainly on the use of elements and attributes and avoiding what might be considered the more "esoteric" capabilities of XML (such as entities, notations, and so on). This is the domain of XML as a **data interchange format** rather than as a **document markup** language (which was the original driving force behind the introduction of XML).

An **XML View** is a mapping of a data source, which typically is in a non-XML format, through to an XML document—such as mapping a set of relational tables from a database. These XML Views virtualize the data, in that it isn't actually converted into an XML (serialized) format but merely shaped and transformed into a structure as if it were XML, based on the XML InfoSet data types. The use of XML Views over relational databases allows generation of XML documents and the hierarchical data structure, and also enables the use of XML technologies such as XQuery to provide heterogeneous queries over disparate data sources.

XML is also a more flexible data format than its relational equivalent because it has the ability to express the **semi-structured** and **unstructured** format that applies to much of the data lying outside of relational databases. Semi-structured refers, for example, to properties that appear only on a certain number of elements and not in a regular repeating (structured) fashion. Unstructured format is like that of a Microsoft Word document, typically consisting of "marked-up" data such as the content of the paragraphs in the text. XML has both the simplicity and flexibility to easily represent these differing data structures, and as a result XML has established itself as the lingua franca of data interchange for business and applications.

XML in Web Applications

XML as a data format is well suited to the environment in which Web-based applications live, being a platform-independent syntax for which simple and efficient parsers are widely available. In particular, any situation that involves remoting data (moving it across the network to a client or another server, as we describe in more detail later) can benefit from the use of XML. For example, clients or servers that run on disparate operating systems or software platforms can share XML-formatted data easily. Being persisted as Unicode means that even a 7-bit wire format (such as the "text-only" nature of the Web) for the intervening network does not impede simple and efficient data transmission.

In contrast to this, application- or operating system–specific data formats, particularly binary formats, may well cause all kinds of issues when used with disparate clients. As a simple example, some operating systems may treat 16-bit numbers as big-endian (the leftmost byte, at the lower memory address, is the most significant) while others treat them as little-endian (the leftmost byte is the least significant).

XML is also the ideal format for simply sending data to a client or receiving it from a client. For example, the Web site or application can provide a browser client with an XML document that is downloaded to the user's browser. Then it can be manipulated within the browser to display the data and (if required) edited and submitted back to the server for subsequent processing. Because the data is in a platform- and operating system–independent format, the code or application at either end of the network can be adapted and changed without reference to changes at the other end, as long as the data format remains the same.

Data Description through Schemas

Even more useful is the ability to use an XML schema with the XML data. A schema specifies the structure and allowable content of the XML document. This permits applications to be built where the client or recipient uses the schema to interpret the structure of the data, with the result that this data format can be changed independently as long as an appropriate schema is available.

An example of this approach is found in the new XML-format metabase used by Internet Information Services (IIS) 6.0. Each time the configuration of IIS is changed, an XML file containing the complete configuration of all the installed services is written to disk. However, as you install and remove services or add sites or virtual roots to IIS, the structure of this file changes in subtle ways to reflect the content it will store.

So, as well as writing the XML data file to disk, IIS also writes the current schema to disk. This means that, irrespective of which services are installed and how the current configuration looks, the saved configuration data can be correctly interpreted and restored each time.

Transformation and Presentation through XSLT Stylesheets

The concept of applying presentation information to an XML document has been one of the core aims of W3C almost since the inception of XML. The original proposals concentrated on two aspects: (1) **transformations,** which can change the content, structure, and ordering of the XML (to produce, for example, HTML output), and (2) the definition of a **formatting language** used by specialist client applications to apply style and layout information directly to the XML content.

In data management terms, the first of these two scenarios is the interesting one. The ability to apply a transformation that optionally can change the content, structure, and ordering of the elements means that an XSL or XSLT stylesheet can not only generate output that is aimed at presentation (for example, adding style definitions or HTML elements to the content) but also output a different XML document from that applied as the input.

XSLT is increasingly being used to "process" XML documents. Its recursive nature and reasonably simple syntax mean that it's possible to write generic stylesheets that can process different documents, and it has found a home in many areas and applications that need to convert XML data from one format to another (for example, Microsoft BizTalk).

XML Data Querying through XQuery

One of the more recent areas of development for XML is **XML Query Language**, or **XQuery**. This is a project aimed at deriving a technique for applying queries to an XML document that (in the broadest sense) more closely resemble SQL statements. While SQL is ideal as the query language for relational data, it is not designed to be a query language for XML data, since a query language should have as little impedance mismatch to the data model it queries as possible. As mentioned earlier, XQuery is to the XML data model what SQL is to the relational data model and is set to become the preferred query language for XML data querying. This will make XML much more approachable to those developers and administrators more used to working in a relational data environment.

Although XSLT can already perform most of the tasks that XQuery is aimed at, it's already becoming clear that XQuery can provide more options, as well as broader and simpler techniques for accessing multiple documents, than XSLT does. However, the choice between the two is more likely in the long term to be based on the developer's grasp of the techniques and language choice, in an identical manner to the language choice available in the .NET Framework today.

In a nutshell, XQuery provides support for:

- A more readable, SQL-like syntax designed to support query expressions across several XML data sources
- Type checking so that, for example, all the `Order` types defined in an XML schema can be found in an XML document
- A set of functions and operators defined for the built-in XML Schema (XSD) types (for example, the result of adding an XSD integer type to an XSD string type)

The current recommendations and guides to all the specifications we've mentioned here (XML, XSLT, XQuery, and XML Schema) can be found at the W3C Web site at **http://www.w3.org**.

XML Content Publishing

While Web applications are a major beneficiary of XML, they are not alone. Another arena where XML is growing is in content publishing. This is a wide and difficult area to define exactly, but in general the term applies to application and service integration that is aimed at exposing data or information to users.

Microsoft's own Content Management Server is an example, including features to build and deploy Web sites and Web Services, manage updates, perform workflow management, and integrate with Microsoft Office applications. It uses XML to transport and expose information in conjunction with templates, and to package content objects.

Similarly, SharePoint Server and Microsoft Office 2003 contain many features based on XML. In fact, XML is at the forefront of integration in Office 2003, with the new InfoPath data collection application storing data as XML and exposing it to the other "traditional" Office applications such as Word and Excel through Smart Documents based on XML.

XML in the .NET Framework

XML has been at the heart of the .NET Framework since version 1.0. Many aspects of data persistence, storage, and transmission depend wholly or partly on XML. For example, while the ADO `Recordset` object provided a custom binary format by default for persisted data, XML is the only data persistence format supported by the ADO.NET `DataSet`. The `GetXml` and `WriteXml` methods always return XML-formatted data.

Conversions between Relational and XML Data

Figure 1.1 demonstrates some of the ways that XML is used within the .NET Framework for access to and conversion of relational data. The ADO.NET objects `SqlCommand` and `DataSet` can export or expose their content as XML, and the `DataSet` can also export the matching schema.

You can also see that the `XmlDataDocument` can be used to expose XML from relational data, using an existing `DataSet` as the source of the `XmlDataDocument` instance. What's new and exciting in version 2.0, however, is shown in the lower section of the schematic. These are the new features of the `XPathDocument` and its associated new classes. An `XmlAdapter` provides the link between the `XPathDocument` and a relational database. The `XPathChangeNavigator` and `XPathEditor` are used to navigate

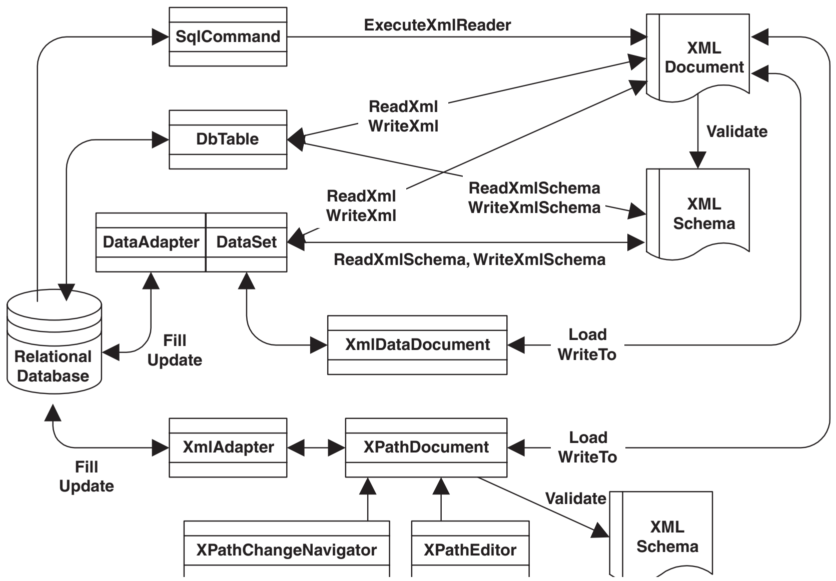


FIGURE 1.1. Access to and conversion of relational data as XML

through and update the XML content, after which the `XmlAdapter` can push the changes back into the database.

XML Serialization and Remoting

The .NET Framework also includes a class called `XmlSerializer` (see Figure 1.2) that can be used to serialize objects and class instances to XML—including custom classes you declare—and deserialize them from XML. This makes it easy to build applications that interchange data represented as objects, such as a purchase order or invoice object generated from custom classes.

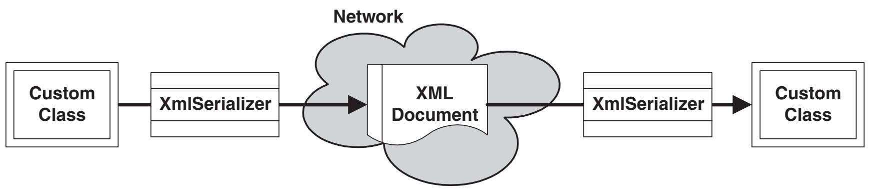


FIGURE 1.2. Serialization of XML data over a network

XML serialization is useful when you need to remote objects (rather than just plain data) to another location, or even just to another tier of your application. However, if you want to remote only the data itself, another useful feature of the XML classes is that the `XmlDataDocument` can be used as an alternative to a `DataSet`. It will maintain all of the schema information required to resurrect the data after transmission — including the data types, relations, and so on. The `XmlDataDocument` can be filled directly with XML of the required format, which can include nested or related data. Alternatively, it can be instantiated from an existing `DataSet`, as long as a schema is available.

However, as mentioned earlier, the `XPathDocument` can also be remoted. Because it implements change tracking, the remoting format has to be able to persist the changes as well as the "current" values of nodes in the document in case this information is required to be persisted in an `XPathDocument` regenerated from the serialized data. Three scenarios are supported.

1. Save or serialize just the data content, not details of the changes made to the values.
2. Save or serialize all the data and all details of the changes made to the values.
3. Save or serialize only the values where changes have occurred, not the "unchanged" values.

The `XPathDocument` serializes values at `Node` level. In other words, every node in the document gets serialized and is then available when the document is rebuilt. In an XML document, every element and attribute is a node, so the complete data content of the document is maintained.

Serialization is actually useful not only when remoting data but also for supporting off-line data access and editing. After using an `XmlAdapter` to fill an `XPathDocument`, you can save the data to a local disk and disconnect from the database, then work with the data without maintaining a connection. Later you can reconnect and resynchronize the data in the database using the changes in the `XPathDocument` in much the same way as you would in the relational world.

Migration of the SQLXML 3.0 Technology to .NET

One of the aims of version 2.0 of the .NET Framework is to move the existing SQLXML technology to the .NET platform. SQLXML was originally developed as an add-on for SQL Server 7.0. The aim was to make it easy to extract data from SQL Server as XML and to use specially formatted XML updategrams to push data updates back into SQL Server.

The original Technology Preview of SQLXML involved installing a "filter" that sits in SQL Server and responds to queries that contain the `FOR XML` keywords. SQL Server 2000 has the SQLXML version 3.0 technology built in, and it automatically detects and processes SQL statements that are XML queries. For example, the following SQL statement:

```
SELECT * FROM Country WHERE Name LIKE 'U%' FOR XML AUTO
```

returns a series of XML elements that represent each matching row in the database table, with each column represented as an attribute of the row:

```
<Country Name="USA" Continent="America" Size="Very large"/>
<Country Name="UK" Continent="Europe" Size="Quite small" />
... etc ...
```

These elements can easily be wrapped in a suitable preamble and root element to turn them into an XML document. In .NET, SQLXML queries that return XML elements are handled by an `XmlReader` returned from a call to the `ExecuteXmlReader` method of a `Command` object connected to the database.

However, it would be useful to be able to use this type of data query, and return XML, with databases other than SQL Server and to make it more compatible with the techniques used in the Framework. This is achieved in version 2.0 of the .NET Framework through the `XmlAdapter` and `XPathDocument` you saw in Figure 1.1, though in the current release this is limited to the SQL Server "Yukon" version. Instead of having to learn complex rules for specifying the format of the data returned from the database and all the different options available for updategrams, you use an approach and a syntax that are similar to those used in relational data access through the ASP.NET `DataAdapter`.

Server-Side Data Binding to XML

In version 2.0, the .NET Framework also supports server-side UI data binding in Web Forms (ASP.NET) and Windows Forms (executable) applications to XML documents stored in the `XPathDocument` and `XmlDocument` classes. This provides a fast and efficient technique for displaying data that is persisted as XML in Web pages and Web applications and in your .NET executable programs.

Relational Data Binding in ASP.NET

One of the great features introduced into ASP.NET in version 1.0 was server-side data binding, through a range of ASP.NET server controls and to many different kinds of data structures. This includes the ADO.NET `DataTable` and `DataReader`, as well as other collection-based structures.

In version 2.0 of ASP.NET, this concept has been extended to allow developers to build data-bound pages that no longer require any code to be written—even for complex operations such as paging, sorting, and updating the original data.

ASP.NET server controls expose a property named `DataSource` to which you assign the source data structure. For example, if you have an active instance of a `DataReader`, you simply assign it to the `DataSource` property of the control and then call the `DataBind` method (on the control or on the `Page` object that contains it) to cause the control to iterate through the data rows and generate the output in the page:

```
MyDataGrid.DataSource = MyDataReader
MyDataGrid.DataBind()
```

If the data source contains more than one table, for example, when using a `DataSet`, the `DataMember` property of the control is used to specify which table is to be bound:

```
MyDataGrid.DataSource = MyDataReader
MyDataGrid.DataMember = "TableName"
MyDataGrid.DataBind()
```

ASP.NET Data Source Controls and the GridView Control

In version 2.0, ASP.NET introduces a range of new server controls to handle data management and server-side data binding. These controls fall into two distinct groups:

1. Data source controls that connect to a data source and expose the data
2. The new `GridView` and `DetailsView` controls that consume the data and display it in a Web page

Several data source controls are included with ASP.NET, all residing in the `System.Web.UI.WebControls` namespace. These include controls designed to work with relational data from SQL Server, OLE-DB and ODBC databases, Microsoft Access, and Oracle. There is also a special data source control designed to work with the new ObjectSpaces technology to enable server-side data binding to custom business objects. Some data source controls consume XML from a stream or disk file. The XML Data Source Controls and Data Binding section in Chapter 7 briefly discusses the controls that consume XML, and the companion book to this one, *A First Look at ASP.NET v. 2.0* (Boston, MA: Addison-Wesley, 2004, ISBN 0-321-22896-0), contains full details of all the data source controls.

The new ASP.NET `GridView` and `DetailsView` controls expose `DataSource` and `DataMember` properties, just like most other "bindable" server controls. However, they also expose a `DataSourceID` property, which can be set to the ID of a data source control that exposes data in relational (columns and rows) format. When this is done, the two controls work together to extract the data and perform the server-side data binding automatically, with no code required.

You'll also be pleased to know that all the ASP.NET server controls that support data binding now expose the `DataSourceID` attribute in version 2.0, so they can be used interchangeably with any data source control that exposes the data in the appropriate format.

An Example of the Data Source and GridView Controls

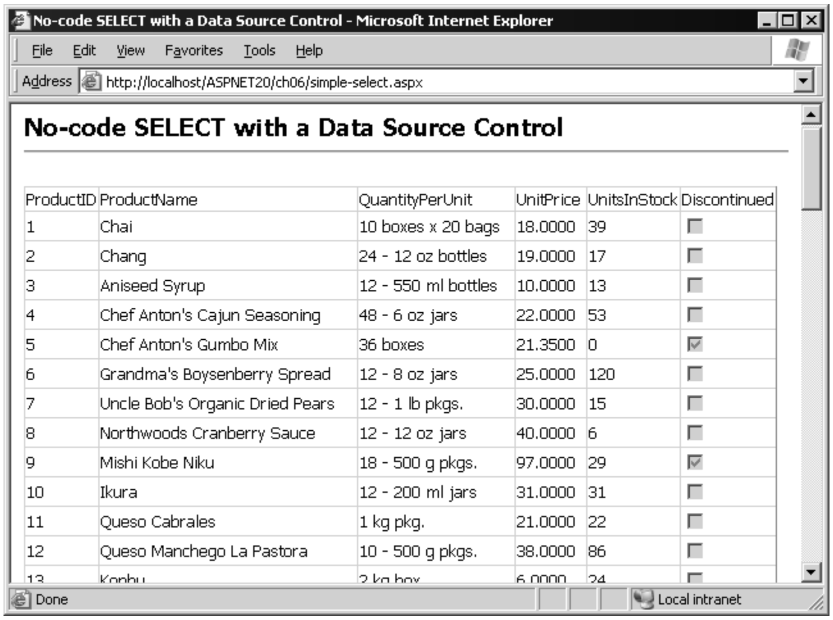
As an example of how these two controls change the whole approach to server-side data binding, the code in Listing 1.1 (taken from the book referred to above) shows the complete `<form>` section of an ASP.NET page that extracts some rows from the Northwind database and displays it in a `GridView` control.

LISTING 1.1. Using a Data Source Control and a GridView Control

```
<form runat="server">

<asp:SqlDataSource id="ds1" runat="server"
  ConnectionString="server=localhost;database=Northwind;uid=x;pwd=x"
  SelectCommand="SELECT ProductID, ProductName, QuantityPerUnit,
                 UnitPrice, UnitsInStock, Discontinued FROM Products"
/>

<asp:GridView id="grid1" DataSourceID="ds1" runat="server" />

</form>
```

The `SqlDataSource` control uses a connection string and a SQL statement (it could alternatively be a stored procedure name), plus "sensible" default settings, to connect to a database and extract the data rows. It reacts to the page-level events that occur when the page is requested and internally builds the usual ADO.NET objects it needs (`Connection`, `DataAdapter`, `DataSet`, and so on).

The `GridView` control performs the data binding and displays the data. The screenshot in Figure 1.3 shows the result. It displays all the rows from the Products table in the Northwind database, just as specified in the SQL statement. However, notice the Discontinued column. This is a `Boolean` field in the table, and the control automatically displays it using read-only (disabled) checkboxes.


No-code SELECT with a Data Source Control - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/ASPNET20/ch06/simple-select.aspx
Done
Local intranet

No-code SELECT with a Data Source Control

ProductID	ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	Discontinued
1	Chai	10 boxes x 20 bags	18.0000	39	☐
2	Chang	24 - 12 oz bottles	19.0000	17	☐
3	Aniseed Syrup	12 - 550 ml bottles	10.0000	13	☐
4	Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.0000	53	☐
5	Chef Anton's Gumbo Mix	36 boxes	21.3500	0	☑
6	Grandma's Boysenberry Spread	12 - 8 oz jars	25.0000	120	☐
7	Uncle Bob's Organic Dried Pears	12 - 1 lb pkgs.	30.0000	15	☐
8	Northwoods Cranberry Sauce	12 - 12 oz jars	40.0000	6	☐
9	Mishi Kobe Niku	18 - 500 g pkgs.	97.0000	29	☑
10	Ikura	12 - 200 ml jars	31.0000	31	☐
11	Queso Cabrales	1 kg pkg.	21.0000	22	☐
12	Queso Manchego La Pastora	10 - 500 g pkgs.	38.0000	86	☐
13	Konbu	2 kg box	6.0000	24	☐

FIGURE 1.3. A data source and GridView control in action

Do I Still Need to Write ADO.NET Code?

In ASP.NET, you may be able to avoid writing code to interact with a database and just rely on the data source controls to do all the work. However, that doesn't mean you can stop reading now! As is always the case, knowing more about how the data access technologies actually work, as well as being able to take advantage of the features they offer to tailor performance and behavior to suit your requirements, is part of the development process.

In particular, you may prefer to take advantage of specific new features in ADO.NET such as MARS, asynchronous processing, and batch command execution to access and expose your data in exactly the way you require. In this case, having generated your data structure, you can continue to benefit from server-side data binding by using the new or existing controls through their `DataSource` and `DataMember` properties.

A Roadmap for This Book

The remainder of this book is divided into two sections, along the same lines as the feature summaries you saw earlier in this chapter. In the next three chapters, we look in more detail at the new features in ADO.NET version 2.0.

- **Chapter 2**, Bulk Loading, Batch Execution, and Paging, looks at the new classes, methods, and other additions to ADO.NET in version 2.0 that enable some exciting new techniques to be applied. This includes bulk inserts and updates in SQL Server, batched execution of commands, and data paging support.
- **Chapter 3**, MARS, Asynchronous Commands, and ObjectSpaces, looks at three more of the new core data access techniques and features added to ADO.NET 2.0. MARS provides the opportunity to open more than one result set over the same connection and access them concurrently, while asynchronous execution of multiple commands over the same `Connection` instance is also possible now. This chapter also introduces the new ObjectSpaces technology added to ADO.NET, which allows data to be defined as instances of a class but handled just like other intrinsic data types.
- **Chapter 4**, ADO.NET and SQL Server “Yukon,” looks at ADO.NET integration with the new version of Microsoft SQL Server (codenamed “Yukon”). It examines notifications, managed code within SQL Server, user-defined data types, and the new feature that allows server-side cursors to be used to directly access and modify data.

Then, in the final four chapters, we look at the changes in the `System.Xml` and subsidiary namespaces.

- **Chapter 5**, New Features of System.Xml, explores and overviews the main basic concepts—such as XML programming and the XML data model, disconnected XML database access, XML Views, XQuery—and how to get started using them.

- **Chapter 6**, The XPathDocument2 Class, concentrates in more depth on the new features of the updated `XPathDocument` class and its associated classes, such as `XPathChangeNavigator` and `XPathEditor`, that provide read/write access to XML documents.
- **Chapter 7**, The XmlAdapter and SqlXml Classes, is a detailed look at the `XmlAdapter` class that, together with the `XPathDocument`, enables disconnected data manipulation in XML. It also covers XML schemas and the three-part mapping technology that links relational and XML data. This chapter also looks at the .NET replacements for the unmanaged SQLXML technology implemented in SQL Server 2000.
- Finally, **Chapter 8**, XQuery and SQL Server, looks at what XQuery is, and the support for it in version 2.0 of the .NET Framework. This includes examination of the XQuery architecture, the new `XQueryProcessor` class, and query composition. This chapter also looks at the new mapping format for defining XML Views over data sources that enable queries and updates and briefly discusses the XML data type in SQL Server "Yukon."

SUMMARY

This short first chapter aims to get you familiar with the overall enhancements to version 2.0 of the .NET Framework over version 1.0 that concern data management using ADO.NET and the classes from the `System.Xml` and subsidiary namespaces. As well as summarizing these as simple lists, we've attempted to fill out the picture a little by including some background on why they have come about and Microsoft's aims as far as developers are concerned.

We also spent some time looking at the area of XML data management within the .NET Framework, to make sure you appreciate the growing importance and widening acceptance of XML as a data persistence, manipulation, and transmission format. As new business applications and software suites appear (such as Microsoft Office System, SharePoint, and so on), the use of and dependence on XML can only continue to grow.

We then briefly overviewed the way that XML is managed within .NET, in both versions 1.0 and 2.0, to reinforce the direction in which development of the Framework is moving. The concept of relational and XML data as being two different entities is rapidly disappearing, and the forthcoming opportunities to construct XML data and schema repositories based on SQL Server will only help to accelerate this change.

Finally, after a quick look at how the changes to ADO.NET affect (or, in fact, do not affect) ASP.NET data binding, we ended the chapter with a brief outline of the content in the remainder of this book. This should help you to dive in and see the areas that are of particular interest to you.

As we stated earlier, the Technology Preview release of the .NET Framework version 2.0 is by no means a complete view of all the features that will be present in the final release. One area that Microsoft is working on, but which has not been finalised yet, is a new class that simplifies relational data management. Early prototypes indicate that it will remove the need for developers to explicitly create intermediate class instances such as Connection, Command, and DataAdapter by using sensible default values and internal heuristics. This feature, providing characteristics similar to but extending the existing DataTable class, should find its way into forthcoming Beta releases of the Framework.