XML and Web Publishing

In This Chapter

Comparing XML and EDI

How XML is used

Understanding XML documents

Validating XML data with DTDs and XSDs

Creating and reading XML data with SQL Server

Creating HTML with XSL style sheets

Using HTTP queries

Generating HTML pages from SQL Server

The Internet is moving from being a vehicle primarily for e-mail and data presentation into a phase of high-performance data connectivity. HTML, as a common mark-up language, describes how data should be presented, but eXtensible Markup Language, or XML, goes deeper than HTML and describes the content of the document.

Mark-up languages began in 1969 when Charles F. Goldfarb, Ed Mosher, and Ray Lorie developed Generalized Markup Language (GML) for IBM, to standardize documents for publication. In 1974, Charles Goldfarb published Standard Generalized Markup Language (SGML), which became "the International Standard (ISO 8879) language for structured data and document representation" (`HTTP://www.sgmlsource.com`). Basically, SGML defines the rules for creating mark-up languages. As a presentation mark-up language that follows the SGML rules, HTML formats the look of data for Web browsers. XML is a subset of SGML that essentially uses ’60s technology to create a new mark-up language that describes a set of data. In 1998, the World Wide Web Consortium (`HTTP://www.W3C.org`) published version 1.0 of the official XML specs.

By itself, XML does nothing; it's only a file format. There is no such thing as an XML application — only applications that read or generate XML documents. However, the wide acceptance of XML makes it an excellent format for transporting data between dissimilar systems, especially over the Internet. The prosaic comma-delimited text file usually identifies the fields in the first line, and, more often than not, the data does not follow the format rule, making comma-delimited files frustrating. As a welcome relief, XML explicitly tags every individual piece of data, reducing the errors caused by varying or incomplete data schemas.

XML uses a hierarchical data structure: each XML element may contain a nested XML element. For example, a customer element that describes a single customer might contain a few order elements, which could contain order-detail elements. Every XML document uses this top-down hierarchal method to contain data. This makes XML very powerful for sharing data sets, but prevents XML from being a replacement for a relational-database system.

On the CD
While the SQL Server–specific code is in the chapter code script as usual, this chapter also refers to several XML document and validation files located on the CD in the `C:\SQLServerBible\Sample Databases\CapeHatterasAdventures` directory. All the XML samples are pulled from the Cape Hatteras Adventures sample database.

In the sample XML document `CHA2_Events.xml`, data extracted from the Cape Hatteras Adventures sample database is painstakingly identified using XML tags. This allows the data to be self-describing and represented in a format that is identical to its representation within the database.

```
<?xml version="1.0" encoding="UTF-8"?>
<Tours>
  <Tour Name="Amazon Trek">
    <Event Code="01-003" DateBegin="2001-03-16T00:00:00"/>
    <Event Code="01-015" DateBegin="2001-11-05T00:00:00"/>
  </Tour>
  <Tour Name="Appalachian Trail">
    <Event Code="01-005" DateBegin="2001-06-25T00:00:00"/>
    <Event Code="01-008" DateBegin="2001-07-14T00:00:00"/>
    <Event Code="01-010" DateBegin="2001-08-14T00:00:00"/>
  </Tour>
  <Tour Name="Bahamas Dive">
    <Event Code="01-002" DateBegin="2001-05-09T00:00:00"/>
    <Event Code="01-006" DateBegin="2001-07-03T00:00:00"/>
    <Event Code="01-009" DateBegin="2001-08-12T00:00:00"/>
  </Tour>
  <Tour Name="Gauley River Rafting">
    <Event Code="01-012" DateBegin="2001-09-14T00:00:00"/>
    <Event Code="01-013" DateBegin="2001-09-15T00:00:00"/>
  </Tour>
  <Tour Name="Outer Banks Lighthouses">
    <Event Code="01-001" DateBegin="2001-02-02T00:00:00"/>
    <Event Code="01-004" DateBegin="2001-06-06T00:00:00"/>
    <Event Code="01-007" DateBegin="2001-07-03T00:00:00"/>
    <Event Code="01-011" DateBegin="2001-08-17T00:00:00"/>
    <Event Code="01-014" DateBegin="2001-10-03T00:00:00"/>
    <Event Code="01-016" DateBegin="2001-11-16T00:00:00"/>
  </Tour>
</Tours>
```

SQL Server 2000 has first-generation XML capabilities to publish to the Web and work with XML data. XML and .NET Web services easily fill a couple more books. The purpose of this chapter is to provide an understanding of how XML fits into the database world, as well as the commonly used features of XML as used by SQL Server.

XML and EDI

XML is the best means of moving data among various dissimilar databases and programs. As such, it is replacing the expensive Electronic Data Interchange (EDI) systems of the ’80s and ’90s and is the new B2B data-exchange standard. The issue with XML is that both companies sharing data must agree on the XML schema or data-descriptor tags. Most industries have already established XML standards.

The Commercial XML standard (`HTTP://www.cXML.org`) has taken the lead as the universal transactional B2B XML standard. cXML is a set of XML Document Type Definitions (DTDs) that define the structure of XML mark-up languages for handling common B2B transactions such as catalogs, product suppliers, punch outs (instances of interactive procurement over the Internet), master agreements, purchase orders, order confirmations, ship notices, and invoices.

Other XML standards include:

* XMLLife — For insurance companies
* Channel Definition Format (CDF) — For pushing media over the Web
* Open Financial eXchange (OFX) — For the stock market and financial industry
* Mathematical Markup Language (MathML) — For describing equations
* Chemical Markup Language (CML) — For chemical formulas and products

In addition, several server software packages are already on the market that convert one XML schema to another for data sharing, including Microsoft BizTalk.

Working with XML

Like an HTML document, an XML document is just a text document with tags — often a very long document with an abundance of tags. While XML documents can be viewed and created using Notepad, XML is not intended for human consumption. Most developers who work with XML use a parser to decode the XML within an application and an XML viewer to peruse the XML document.

XML Parsing

To work with XML data from within custom application programs, you'll want to use a pre-written parser. A parser will decode the XML document and handle the complexities of XML. Several vendors offer XML parsers; Microsoft's parser, XML Core Services V4.0 (formerly known as MSXML), is one of the more complete parsers available and is a free download from:

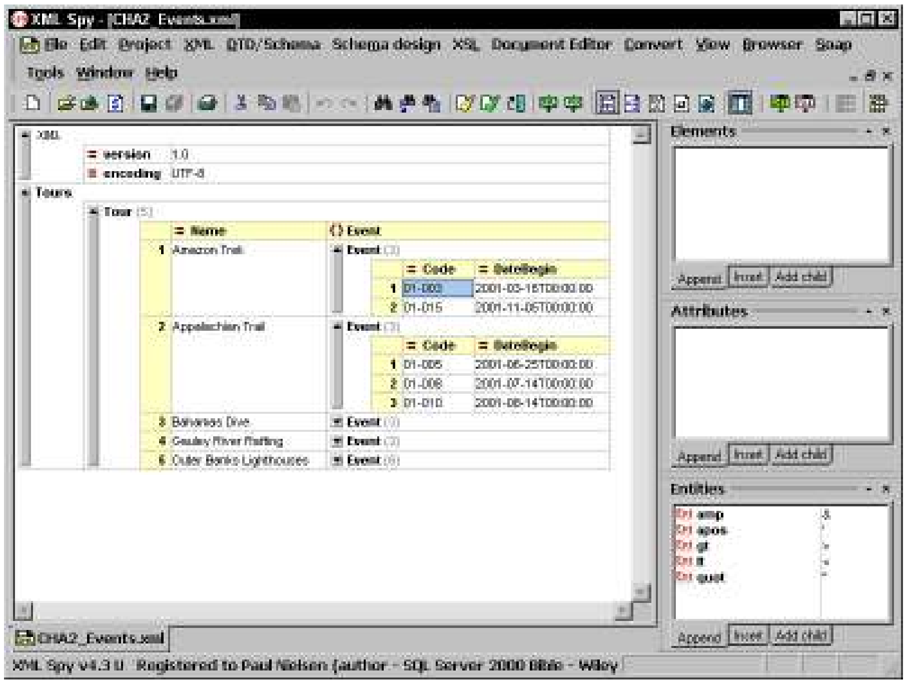
```
HTTP://msdn.microsoft.com/downloads/default.asp?
  url=/downloads/topic.asp?
  url=/MSDN-FILES/028/000/013/topic.xml
```

Within a custom application, code can access the XML Core Services objects to inspect and create XML documents. The Document Object Model (DOM) is the standard API for XML documents. Simple API for XML (SAX) is an alternative to DOM. Microsoft Core Services includes:

* An XML parser
* An XSL style-sheet processor
* Support for DOM

XML Viewing

A number of tools can aid in the viewing and creation of raw XML documents. I highly recommend XML Spy, shown in Figure 22-1, because it's a full suite of XML tools in a slick integrated development environment (IDE) that includes many views and features. It can graphically create or display XMLs, XSLs, DTDs, and XML Schemas, and can validate XML documents.


XML Spy - [CHA2_Events.xml]
File Edit Project XML DTD/Schema Schema design XSL Document Editor Convert View Browser Soap
Tools Window Help
Elements
Append Insert Add child
Attributes
Append Insert Add child
Entities
Append Insert Add child
CHA2_Events.xml
XML Spy v4.3 U Registered to Paul Nielsen (author - SQL Server 2000 Bible - Wiley

```
XML
    = version    1.0
    = encoding   UTF-8
Tours
    Tour (5)
            = Name                      () Event
        1   Amazon Trail                Event (2)
                                                = Code    = DateBegin
                                            1   01-003    2001-03-16T00:00:00
                                            2   01-015    2001-11-05T00:00:00
        2   Appalachian Trail           Event (3)
                                                = Code    = DateBegin
                                            1   01-005    2001-06-25T00:00:00
                                            2   01-008    2001-07-14T00:00:00
                                            3   01-010    2001-08-14T00:00:00
        3   Bahamas Dive                Event (3)
        4   Gauley River Rafting        Event (2)
        5   Outer Banks Lighthouses     Event (6)
```

```
amp   &
apos  '
gt    >
lt    <
quot  "
```

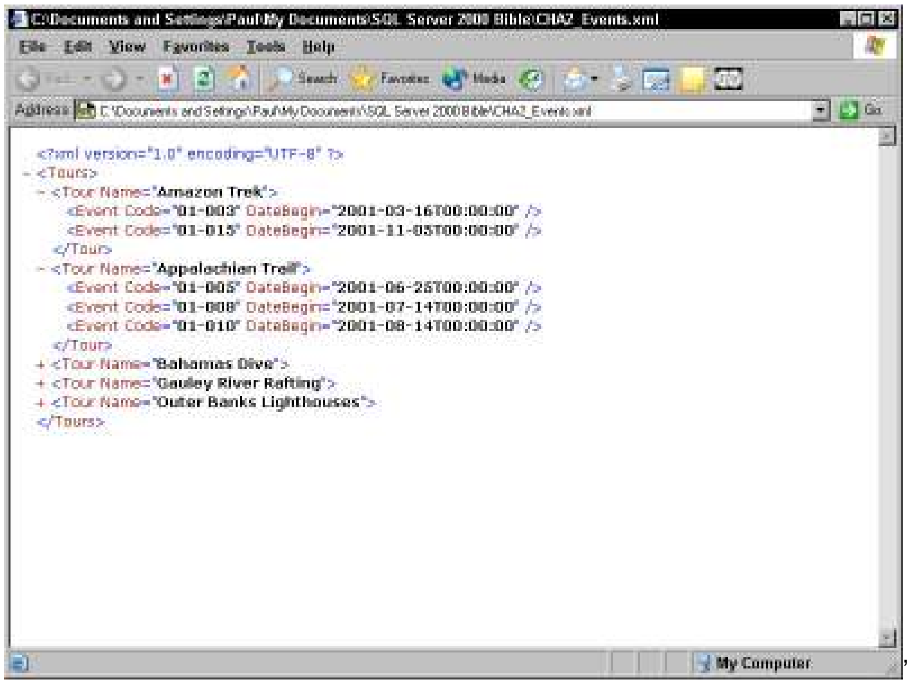
insert fg2201.jpg

Figure 22-1: XML Spy's grid view makes it easy to visualize the elements and attributes of the sample CHA_Event.xml document.

On The CD

XML Spy is the best tool for working with XML documents. A 30-day trial version of XML Spy is on the CD included with this book. XML Spy includes several example documents.

An XML document can be opened and viewed within Internet Explorer. IE 5 includes a default XML style sheet (Figure 22-2) for browsing the XML data.


C:\Documents and Settings\Paul\My Documents\SQL Server 2000 Bible\CHA2_Events.xml
File Edit View Favorites Tools Help
Search Favorites Media
Address C:\Documents and Settings\Paul\My Documents\SQL Server 2000 Bible\CHA2_Events.xml
Go
My Computer

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Tours>
  - <Tour Name="Amazon Trek">
      <Event Code="01-003" DateBegin="2001-03-16T00:00:00" />
      <Event Code="01-015" DateBegin="2001-11-05T00:00:00" />
    </Tour>
  - <Tour Name="Appalachian Trail">
      <Event Code="01-005" DateBegin="2001-06-25T00:00:00" />
      <Event Code="01-008" DateBegin="2001-07-14T00:00:00" />
      <Event Code="01-010" DateBegin="2001-08-14T00:00:00" />
    </Tour>
  + <Tour Name="Bahamas Dive">
  + <Tour Name="Gauley River Rafting">
  + <Tour Name="Outer Banks Lighthouses">
  </Tours>
```

'insert fg2202.jpg

Figure 22-2: IE 5 provides a cool collapsible-list feature, which is not a standard XML view but is actually the default XSL style sheet within IE 5.

XML Publishing

Because XML describes the content of the data, various style sheets can locate data within the XML document and format it for various devices. A single XML document might be viewed using any of the following:

* *Computer browsers using XSL* — XML style sheets that merge XML data into an HTML document
* *Cell phones or other wireless devices using Wireless Markup Language (WML)* — The following link provides details on the Wireless Application Protocol specifications:

  `www.wapforum.org/what/technical.htm`

* *Acrobat Reader – XML:FOP* — IBM's Formatting Objects to PDF tool can use XML and XSL to produce PDF files.

  ```
  HTTP://www-106.ibm.com/developerworks/education
    /transforming-xml/xmltopdf/

  HTTP://www.wapforum.org/what/technical.htm
  ```

A major component of Microsoft's .Net initiative is Web services, which reply to queries with XML data. Combining Web services with XSL is an excellent way to publish dynamic data on the Web.

XML Validation

A key benefit of XML is its ability to validate data using Data Type Documents or XML Schema documents. These additional documents describe the data structure so the receiving application can compare the data to the expected structure. The validation information can be stored inside the XML document, or the XML document can point to the validation document.

You can use the validation and publishing features of XML to develop powerful Web-based applications.

Inside an XML Document

An XML document consists of two sections: the Declaration section, which contains information for the XML parser, and the Root-Element section, which contains the data.

As a database developer, you won't spend your days writing XML documents by hand; instead a tool or SQL Server will generate the documents.

Declaration Section

The Declaration section is an optional prologue to the document and contains Processing Instructions (PI). Typically, the Declaration section specifies instructions, such as the following, for the parser:

```
<?procname procatt="attribvalue" ?>
```

The most common instruction identifies the document as an XML document using the reserved PI, XML:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Other instructions can specify the DTD file for data validation, or the XSL file for translation into an HTML document.

Root Element

Each data object in XML is defined as an element. The root element is the overarching object container. All the data contained in an XML document must fit inside elements and attributes contained within a single root element.

In this sense an XML document is no different from an HTML document that uses the `<HTML>` tag as its root element. The XML root name should be a word that describes the content of the XML document, such as `<Orders>`, `<Customers>`, or, as in the case of the sample XML document, `<Events>`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Tours>
  <Tour Name="Amazon Trek">
    <Event Code="01-003" DateBegin="2001-03-16T00:00:00"/>
    <Event Code="01-015" DateBegin="2001-11-05T00:00:00"/>
  </Tour>
  <Tour Name="Appalachian Trail">
    <Event Code="01-005" DateBegin="2001-06-25T00:00:00"/>
    <Event Code="01-008" DateBegin="2001-07-14T00:00:00"/>
    <Event Code="01-010" DateBegin="2001-08-14T00:00:00"/>
  </Tour>
  <Tour Name="Bahamas Dive">
    <Event Code="01-002" DateBegin="2001-05-09T00:00:00"/>
    <Event Code="01-006" DateBegin="2001-07-03T00:00:00"/>
    <Event Code="01-009" DateBegin="2001-08-12T00:00:00"/>
  </Tour>
  <Tour Name="Gauley River Rafting">
    <Event Code="01-012" DateBegin="2001-09-14T00:00:00"/>
    <Event Code="01-013" DateBegin="2001-09-15T00:00:00"/>
  </Tour>
  <Tour Name="Outer Banks Lighthouses">
    <Event Code="01-001" DateBegin="2001-02-02T00:00:00"/>
    <Event Code="01-004" DateBegin="2001-06-06T00:00:00"/>
    <Event Code="01-007" DateBegin="2001-07-03T00:00:00"/>
    <Event Code="01-011" DateBegin="2001-08-17T00:00:00"/>
    <Event Code="01-014" DateBegin="2001-10-03T00:00:00"/>
    <Event Code="01-016" DateBegin="2001-11-16T00:00:00"/>
  </Tour>
</Tours>
```

The element concludes with a closing tag, `</Events>`, as seen in the example above.

Elements

XML is loose in its usage of elements and attributes. An element is generally an item similar to a tuple or row. However, a column can also be represented within XML as an element. Element names cannot contain spaces, must be unique, and are case-sensitive.

Elements can contain the following:

* Other nested elements to represent secondary table information or columns
* Attributes, which are similar to database columns
* Text data

The `element` tag is repeated for each instance of the element. For example, the `CHA2_Events.xml` sample XML document has a series of `Tour` elements beginning with the `<Tour>` tag and closing with the `</Tour>` tag. The first `Tour` element is this:

```
<Tour Name="Amazon Trek">
```

The tag denotes a Tour. In this case, the element has an attribute, `Name`, with a value of `"Amazon Trek".`

If an element contains nested elements, it will close with a closing tag:

```
<Tour Name="Amazon Trek">
  Other elements
</Tour>
```

If the element does not contain any additional elements, it can be self-closed with a slash at the conclusion of the element:

```
<Event Code="01-003" DateBegin="2001-03-16T00:00:00"/>
```

Data that a database developer would consider an attribute is often expressed within an XML document as a valid nested element:

```
<Tour>
  <Name="Amazon Trek">
</Tour>
```

An element may also contain stand-alone text referred to as *PCData* (parsable character data):

```
<Thing "This is a PCData text within an element">
</Thing>
```

Although elements and attributes seem interchangeable, elements can be repeated while attributes cannot. While Java Web developers might consider an element a valid location for a column data, database developers have been describing data for a few decades now, and know better.

Attributes

Database tables have columns and XML documents can have attributes to further describe the element. Attribute names must be unique and contain no spaces. The attribute value is enclosed in single or double quotes. Each attribute may only be applied to an element once. In the following case, name is an attribute describing the tour element:

```
<Tour Name="Amazon Trek">
```

Note
XML lacks an explicit `null` value. An attribute can be an empty string, but `null`s cannot be passed with XML without an explicit `Column-Null` attribute being specified and the attribute being set to `1` to indicate a `null`.

Namespaces

XML can share data between dissimilar systems, and data names are not always unique. For example the term *stock* has a different meaning in the financial, stock-car racing, firearms, and lumber industries. XML uses a namespace to differentiate between similar terms by providing a unique name qualifier. The namespace is a shortcut that refers to a unique string. While any unique string can differentiate namespaces, the standard practice is to use the string to point to the URL location of the Data Type Document for the data associated with the namespace.

In the following example, the `tr` namespace refers to a unique URL string:

```
<?xml version="1.0" encoding="UTF-8"?>
<Tours xmlns:tr="HTTP://www.CHA2.com/tr.dtd">
  <tr:Tour Name="Amazon Trek">
</Tours>
```

Well-Formed XML Documents

XML is very strict concerning formatting and syntax, much stricter than HTML or SQL. HTML browsers have become very liberal in their allowance for poorly formed HTML tags (`<p>` without `</p>`, tags improperly nested, tags with the wrong case, and so forth). The following are the five rules for a well-formed XML document:

1. Each XML document must have one root element.
2. Element or attribute names must not include any spaces.
3. All elements must be correctly nested and terminated:

   ```
   <x> <y> </y> </x>
   ```

4. All tags are case-sensitive.
5. All attribute values must be in quotes.

XMLSpy's yellow checkmark tool tests the current XML document for adherence to the well-formed rules.

XML Text

XML is primarily a text document and as such it has a few well-defined rules concerning text, substituting special characters, and comments.

PCData

Parsable Character Data, or PCData, is any text that the parser will see as it reads the document. Elements must be PCData.

CData

CData (Character Data) is text that the parser will ignore. CData is defined with brackets and the term `CDATA`. The following text is invalid because of the ampersand in the text. Using CData, the parser bypasses the text:

```
<Couple><![CDATA[Paul & Melissa]]</ Couple >
```

Entities

An entity is a predefined shortcut, or placeholder, for a symbol that would otherwise cause a syntax error. For example, angle brackets (`< >`) are illegal because they are used to create tags. The standard entities are listed in Table 22-1:

Table 22-1: Standard Entities

Entity	*Meaning*
`&lt;`	`<`
`&gt;`	`>`
`&amp;`	`&`
`&quot;`	`"`
`&apos;`	`'`

The invalid XML element shown previously can be legally written with an entity:

```
<Couple "Paul &amp; Melissa" />
```

Comments

XML comments are the same as HTML comments, except that XML comments can't include extra hyphens:

```
<!-- Comment  -->
```

Document Type Definitions (DTDs)

An advantage of XML over SQL record sets is that an XML document can include data validation and schema information.

A Document Type Definition is a set of rules that defines the structure of an SGML document. Each SGML document may use only one DTD file. (HTML uses a DTD document that's built into the browser.) XML documents can use custom DTDs. The DTD may be in a separate file or included within the XML document. If the DTD is in a separate file, multiple XML documents can share the same DTD.

An XML document must pass two tests. The first test is whether it's a well-formed XML document that adheres to the five rules. In XML terms, a valid document is one that is well formed and has passed a second data-validation test using its DTD. XML Spy's green check tool performs a validation check.

DTD Structure

The DTD defines the root, elements, repetition of elements, and PCData within the elements. DTD syntax keywords start with a bang (exclamation point) and are uppercase.

Best Practice

I recommend developing DTD for your XML documents as a method of improving data validity. XML Schemas are more powerful than DTDs, but they aren't accepted by as many parsers. For now, DTDs are the standard.

Elements

The following DTD example (`CHA2_Events.DTD`) defines the elements within the sample XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Event EMPTY>
<!ATTLIST Event
   Code CDATA #REQUIRED
   DateBegin CDATA #REQUIRED
>
<!ELEMENT Tour (Event+)>
<!ATTLIST Tour
   Name CDATA #REQUIRED
>
<!ELEMENT Tours (Tour+)>
```

Note

XMLSpy can automatically generate DTDs from existing XML documents and can even create database tables from DTDs.

The `<!ELEMENT` tag defines an element and specifies any nested elements within the element. For example, the `<!ELEMENT Tour (Event+)>` tag indicates that the `Tour` element contains an `Event` element.

Nested Elements

Any nested elements are added to elements in a DTD by being listed within parentheses following the element. The nested element's cardinality (which XML refers to as *repetition*) is defined by the occurrence indicators, as listed in Table 22-2. The occurrence indicator is added to the nested element as a suffix.

Table 22-2: Nested Element Occurrence Indicators

Indicator	*Element Repetition*	*Element Required*

None (Default)	One	Required
+	One to many	Required
?	Zero to one	Optional
*	Zero to many	Optional

In the sample DTD file, the `+` in the root-element definition indicates that the `Events` root element must include one to many `Event` elements:

```
<!ELEMENT Tours (Tour+)>
```

In the following DTD sample line, a `Customer` element may include any number of `order` elements, must include at least one `address` element, and must include exactly one `account` element:

```
<!ELEMENT Customer (Order*, Address+, Account)>
```

Each element definition includes the definition of the next level of nested elements. If the `Order` element includes `OrderDetail` elements, the `OrderDetail` element must be defined within the `Order` element, not the `Customer` element.

Listing the nested elements within parentheses also defines the order of the nested elements. So in the case of the previous `Customer` element, the subelements must be in the following order: `Order` elements (if any), `Address` elements, and `Account` elements.

Nested elements, which may be substituted, are separated by pipe symbols (`|`) rather than commas. The following example indicates that the `MainElement` must contain one `element1` and one or more of either `element2` or `element3`:

```
 MainElement(element1, (element2 | element3)+ )
```

In addition to listing specific elements, `ANY` allows any defined keyword in any order and `EMPTY` allows elements with no data.

General Entities

In addition to the special character entities such as (`&quot;`), general entities may be defined in the DTD and then used in the XML document. A general entity is similar to a constant in a procedural programming language. Parameter entities may be nested. The following line creates a `companyname` entity:

```
<!ENTITY COMPANYNAME "OBX Kites Company">
```

Within the XML document, the general entity will be replaced by the value when the data is read. For example,

```
Company - &COMPANYNAME;
```

becomes

```
Company - OBX Kites Company
```

General entities are useful for building generic XML documents with values that change depending on the DTD document used.

Parameter Entities

A DTD parameter entity (defined with a `%`) is used to insert an external DTD document similar to a `#include` compiler command in procedural languages. The parameter entity must be defined before it's used.

The following example demonstrates including the tour element as a parameter entity. The `tour.dtd` file is inserted into the `event.dtd` document:

```
Tour.dtd file:
<!ELEMENT Tour (Event+)>
<!ATTLIST Tour
   Name CDATA #REQUIRED
>
```

```
Event.dtd file:
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % TOUR SYSTEM "tour.dtd">
<!ELEMENT Event EMPTY>
<!ATTLIST Event
   Code CDATA #REQUIRED
   DateBegin CDATA #REQUIRED
>
%Tour;
<!ELEMENT Tours (Tour+)>
```

Defining Attributes

Within a DTD, an element's attributes may be defined with the `ATTLIST` keyword. In the sample DTD document shown previously, the `Code` and `DateBegin` attributes of the `Event` element are defined in the following tag:

```
<!ATTLIST Event
    Code CDATA #REQUIRED
    DateBegin CDATA #REQUIRED
>
```

XML DTDs don't provide for full-featured data-typing or constraints as database developers would think of constraints. But they can configure the required status of an attribute using the following keywords:

* `#REQUIRED` — An attribute value must be present in the XML document.
* `#IMPLIED` — If the attribute value is missing in the XML document, the application will supply the value.
* `#FIXED` — The DTD's attribute value will be passed as data regardless of the value in the XML document. (The parser should produce an error if the DTD's fixed value is changed in the XML document, but many parsers don't check this.)

Unlike elements, DTDs, XSD (XML Schema Document) don't establish an attribute order.

ID Attributes

The `ID` attribute option defines a unique constraint on the data within the XML document. `ID` is limited by the fact that it cannot accept numeric data, so it's useful only for unique names or codes that include mixed character data.

The following example forces the tour names to be unique:

```
<!ATTLIST Tour
    Name CDATA ID #REQUIRED
>
```

IDREF Attributes

An `IDREF` attribute option is a loose foreign-key reference within the XML document. An attribute defined as an `IDREF` must have data that refers to data that's been validated by an `ID` attribute — any `ID` attribute. Since by the very nature of XML documents (elements nested with elements) most one-to-many relationships are handled, `IDREF` attributes are useful for reflexive relationships.

Referencing the DTD

A DTD is only useful when an XML document references it for validation. An XML document may either include the DTD definition itself within the XML document or reference an external DTD document. The advantage of an external DTD is that it becomes a single point of reference for multiple XML documents.

Internal DTDs are useful for working with a changing schema, such as during development or testing. Each version of an XML document contains its own validation information.

Referencing an External DTD

When the DTD is an external document, it's referenced within the declaration section of an XML document. The location of the DTD document is often an `HTTP` URL (although if you installed the book’s CD the Tours DTD document is located under the `SQLServerBible` directory). The `CHA2_Events_DTDexternal.xml` file includes a reference to an external DTD:

```
<!DOCTYPE Tours SYSTEM
  "C:\SQLServerBible\Sample Databases\CapeHatterasAdventures\
   CHA2_Tours.dtd">
```

Referencing an Internal DTD

To include the DTD validation information inside the XML document, the entire DTD definition is inserted within a `DOCTYPE` tag and square brackets:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Tours [
   DTD definition goes here
]>
<Tours>
  <Tour Name="Amazon Trek">
    <Event Code="01-003" DateBegin="2001-03-16T00:00:00"/>
…
```

The `CHA2_Events_DTDinternal.xml` file includes an internal DTD definition.

XML Schema — XSDs

To database professionals the term *schema* refers to the logical or physical design of the data. Within the Web-oriented XML community, a schema is a specific type of XML validation.

XML Schema Document, or XSD, is an elegant means of performing XML document validation. XML itself is used to define the XML elements. While Data Type Definition documents are the accepted means of XML validation, the newer XML Schema is more powerful and is used within .Net.

* XSDs are themselves XML documents. As such, the familiar XML syntax may be created and edited using standard XML tools. XSDs must be well-formed XML documents.
* From a database viewpoint, an important factor is that XSDs include data typing, something that's lacking in DTDs.
* XSD can include namespace definitions.

XSD was approved as a recommendation by W3C on May 2, 2001, and at the end of 2002 was starting to be used. Be sure to check `HTTP://www.w3c.org` for updates to the XSDs.

On the CD

Altnova's XML Spy will work with XSDs. An XSD within XML Spy is referred to as a W3C schema in the Generate DTD/Schema dialog.

The following is the XSD definition for the previous sample XML document, which contains event data from the CHA2 database generated by XML Spy (`CHA2_Events.XSD`). In the declaration section of the XML document the `XMLSchema` namespace is referenced. Each XML element and attribute (in bold) is defined within with the XSD element structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by
      XML Spy v4.3 U (HTTP://www.xmlspy.com)-->
<xs:schema xmlns:xs
      ="HTTP://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified">
  <xs:element name="Event">
    <xs:complexType>
      <xs:attribute name="Code"
          type="xs:string" use="required"/>
      <xs:attribute name="DateBegin"
          type="xs:dateTime" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Tour">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Event" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="Name"
          type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Tours">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Tour" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schemas are generally much longer than the equivalent DTD definitions. While DTDs are relatively simple, XML Schemas can be very complex.

An XSD document is an XML document that describes an XML document, so an XML element is defined by an `xsd:element` (`xsd` is the XSD namespace). XML Schema includes two types of XML elements, *simple* and *complex*. A simple element only includes text (`PCData`), no subelements or attributes. The XML `Item` element is defined in XSD as a simple element:

```
XML: <Item> Big Kite</Item>
XSD: <xsd:element name="Item" type="xsd:string">
```

Complex elements can contain subelements and attributes and mixed content (PCData and other elements). The previous sample XSD document includes complex elements.

XSD Elements

Elements and attributes defined within the root element are considered global elements and may be referenced later in the XSD document. This adds consistency and reusability to elements within XSD documents. An element, once defined, may be referenced by means of the following syntax:

```
<xsd:element ref="Event"/>
```

Element Cardinality

XSD includes a flexible method of defining the occurrences of a subelement using the `MinOccurs` and `MaxOccurs` element options. The default is `1`; setting the `MaxOccurs` to `unbounded` permits a true one-to-many relationship:

```
<xsd:element name="Event"
   minOcurrs = "0"
   maxOccurs = "unbounded"
```

Element Grouping

XSD has three methods of ordering subelements:

* `<xsd:sequence>` — The XML elements may be in the sequenced order.
* `<xsd:all>` — The XML elements may be used once, in any order.
* `<xsd:choice>` — Any one of the elements may be in the XML document.

The grouping tags (`sequence`, `all`, or `choice`) are wrapped around the subelements.

XSD Attributes

XML Schemas define attributes within an attribute XSD element called `xsd:attribute`. For example, within the sample XSD schema, the `name` attribute is defined by a subelement under the `tour` element:

```
<xs:attribute name="Name"
    type="xs:string" use="required"/>
```

The `use` option may be used to define the attribute as `required` or `optional` (default).

XSD Data Types and Validation

This is the area where XSDs show significant improvement over DTDs. The W3C XML Schema defines over 40 data types, most of which map well with SQL Server.

Beyond the base data types, derived type, much like a user-defined type, may be added by means of using the `simpletype` data type and building data facets to describe the data validation. An XSD facet is a data-validation constraint, similar to a check constraint, that can perform one of several data checks.

Referencing an XSD Schema

An XSD Schema document may be referenced within the root element of an XML document. The following code in the `CHA2_Tours_XSD.xml` document references the `CHA2_Tours.xsd` document (abridged):

```
<?xml version="1.0" encoding="UTF-8"?>
<Tours xmlns:xsi="HTTP://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="C:\SQLServerBible\
   Sample Databases\CapeHatterasAdventures\CHA2_Event.xsd">
<Tour Name="Amazon Trek">
  <Event Code="01-003" DateBegin="2001-03-16T00:00:00"/>
…
```

XML Spy can automatically generate the XSD and reference it within the XML document.

Best Practice

You don't need to decide between DTD and XSD for validation — use both. Because DTDs are more established, I recommend always using DTDs and optionally sending an XSD schema for those data partners who may opt to use it.

XML and SQL Server

SQL Server 2000 enables you to create and read XML data from within a `select` statement.

Creating XML with SQL Server 2000

SQL Server 2000 can produce XML documents directly from queries. The `for XML` optional `select` suffix directs SQL Server to format the query-data result as an XML document rather than as a standard SQL result set.

Note

The XML output appears as a single-column result. By default, the Query Analyzer `maximum size per column` is set too low to view the XML result set. It may be changed to its maximum setting of 8192 in the Results tab of the Options dialog opened from Tools @@> Options.

The `For XML` suffix generates the XML data, but not the declaration section or the `root` element. The application will need to wrap the data from SQL Server correctly to create a result set that qualifies as a well-formed XML document.

For XML Raw

The `For XML` suffix has three modes: `raw`, `auto`, and an `elements` option. The `For XML Raw` mode simply dumps the result-set rows to an XML document without generating any hierarchical structure. Each SQL row becomes an XML `row` element:

```
SELECT Tour.Name, Event.Code, Event.DateBegin
```

```
  FROM Tour
  JOIN Event
    ON Tour.TourID = Event.TourID
  FOR XML RAW
```

Result (abridged):

```
<row Name="Amazon Trek" Code="01-003"
  DateBegin="2001-03-16T00:00:00"/>
<row Name="Amazon Trek" Code="01-015"
  DateBegin="2001-11-05T00:00:00"/>
<row Name="Appalachian Trail" Code="01-005"
  DateBegin="2001-06-25T00:00:00"/>
<row Name="Appalachian Trail" Code="01-008"
  DateBegin="2001-07-14T00:00:00"/>
<row Name="Appalachian Trail" Code="01-010"
  DateBegin="2001-08-14T00:00:00"/>
…
```

For XML Auto

The `auto` mode determines any hierarchies within the data structure and generates a much more useable XML document. The previous sample XML document at the beginning of this chapter was produced with the following query:

```
SELECT Tour.Name, Event.Code, Event.DateBegin
  FROM Tour
  JOIN Event
    ON Tour.TourID = Event.TourID
  FOR XML AUTO
```

The `elements` option causes the `for XML auto` mode to generate elements instead of attributes. The following variation of the sample XML document uses the `elements` option to generate elements exclusively:

```
SELECT Tour.Name, Event.Code, Event.DateBegin
  FROM Tour
  JOIN Event
    ON Tour.TourID = Event.TourID
  FOR XML AUTO, ELEMENTS
```

Results (abridged):

```
<Tour>
  <Name>Amazon Trek</Name>
  <Event>
    <Code>01-003</Code>
    <DateBegin>2001-03-16T00:00:00</DateBegin>
  </Event>
  <Event>
    <Code>01-015</Code>
    <DateBegin>2001-11-05T00:00:00</DateBegin>
  </Event>
</Tour>
<Tour>
  <Name>Appalachian Trail</Name>
  <Event>
    <Code>01-005</Code>
    <DateBegin>2001-06-25T00:00:00</DateBegin>
  </Event>
  <Event>
    <Code>01-008</Code>
    <DateBegin>2001-07-14T00:00:00</DateBegin>
  </Event>
  <Event>
    <Code>01-010</Code>
    <DateBegin>2001-08-14T00:00:00</DateBegin>
  </Event>
</Tour>
…
```

Reading XML into SQL Server

Applications read XML using a parser that in turn exposes the XML data within the Document Object Mode, or DOM. This W3C-established standard is an object-oriented representation of an XML document. The XML document, and each element, attribute, and text within the document, becomes a DOM object. DOM is very powerful and may be used within object-oriented code to create, read, or modify an XML document.

SQL Server uses the Microsoft XML parser and DOM to read an XML document in a two-stage process:

1. The `sp_xml_preparedocument` stored procedure reads the XML document using the MSXML parser and creates the DOM objects internal to SQL Server. The DOM object is identified by an integer returned by the stored procedure.
2. `OpenXML` is used as a data source within an SQL DML statement. `OpenXML` identifies the DOM object using the integer returned from `sp_xml_preparedocument`.

The following code sample first sets the sample XML data into the `@XML` SQL Server local variable. SQL Server then reads data into SQL using the previous two stages, as follows:

1. When the `sp_xml_preparedocument` store procedure is executed, the DOM is created. The DOM ID is received as an output parameter from the stored procedure and stored in the `@iDOM` variable.
2. The `Select` statement refers to the `OpenXML` system function as a data source. It accepts three parameters:
   * The integer ID of the internal DOM object, which was stored in the `@iDOM` variable.
   * The `rowpattern` of the XML document, which `OpenXML` used to identify the element structure of the XML data. In this case the `rowpattern` is `'/Tours/Tour/Event'`.
   * The XML configuration flag, which determines how the elements and attributes are interpreted by `OpenXML` according to Table 22-3.
3. The `OpenXML`'s `With` option forces a column matching for the result set passed back from `OpenXML`. A column is defined by its XML name, data type, and optional XML `element` location.

Table 22-3: OpenXML Configuration Flags

Flag Value	***Setting***	***Description***
0	Default	Defaults to attribute-centric
1	Attribute-centric	`OpenXML` looks for attributes
2	Element-centric	`OpenXML` looks for elements
8	Combined	`Open XML` looks for attributes and then looks for elements

The batch closes with the `sp_removedocument` system stored procedure, which releases the DOM from memory:

```
DECLARE
  @iDOM int,
  @XML VarChar(8000)

Set @XML = '
<?xml version="1.0" encoding="UTF-8"?>
<Tours>
  <Tour Name="Amazon Trek">
```

```
      <Event Code="01-003" DateBegin="2001-03-16T00:00:00"/>
      <Event Code="01-015" DateBegin="2001-11-05T00:00:00"/>
    </Tour>
    <Tour Name="Appalachian Trail">
      <Event Code="01-005" DateBegin="2001-06-25T00:00:00"/>
      <Event Code="01-008" DateBegin="2001-07-14T00:00:00"/>
      <Event Code="01-010" DateBegin="2001-08-14T00:00:00"/>
    </Tour>
    <Tour Name="Bahamas Dive">
      <Event Code="01-002" DateBegin="2001-05-09T00:00:00"/>
      <Event Code="01-006" DateBegin="2001-07-03T00:00:00"/>
      <Event Code="01-009" DateBegin="2001-08-12T00:00:00"/>
    </Tour>
    <Tour Name="Gauley River Rafting">
      <Event Code="01-012" DateBegin="2001-09-14T00:00:00"/>
      <Event Code="01-013" DateBegin="2001-09-15T00:00:00"/>
    </Tour>
    <Tour Name="Outer Banks Lighthouses">
      <Event Code="01-001" DateBegin="2001-02-02T00:00:00"/>
      <Event Code="01-004" DateBegin="2001-06-06T00:00:00"/>
      <Event Code="01-007" DateBegin="2001-07-03T00:00:00"/>
      <Event Code="01-011" DateBegin="2001-08-17T00:00:00"/>
      <Event Code="01-014" DateBegin="2001-10-03T00:00:00"/>
      <Event Code="01-016" DateBegin="2001-11-16T00:00:00"/>
    </Tour>
  </Tours>'

  -- Generate the internal DOM
  EXEC sp_xml_preparedocument @iDOM OUTPUT, @XML

  -- OPENXML provider.
  SELECT *
    FROM OPENXML (@iDOM, '/Tours/Tour/Event',8)
           WITH ([Name] VARCHAR(25) '../@Name',
                 Code VARCHAR(10),
                 DateBegin DATETIME
                )
  EXEC sp_xml_removedocument @iDOM
```

Result (abridged):

```
Name                 Code     DateBegin
-------------------- -------- -------------------------
Amazon Trek          01-003   2001-03-16 00:00:00.000
Amazon Trek          01-015   2001-11-05 00:00:00.000
Appalachian Trail    01-005   2001-06-25 00:00:00.000
Appalachian Trail    01-008   2001-07-14 00:00:00.000
…
```

Transforming XML with XSL

One of the most powerful aspects of XML is the ability to present XML data in different formats defined by XML style sheets or XSL. Typically, the style sheet will wrap HTML around the XML to present a Web page. When an XML document that references an XSL style sheet is opened within a browser, the XSL style sheet becomes a framework HTML page, which is populated from the XML document's data.

Not all XML parsers perform XSL parsing; fortunately, the MSXML parser includes an XSL parser. The processor that performs the XSL transformation is referred to as the XSLT processor.

XSL is not limited to transforming the XML into desktop-computer HTML code. It can transform the data to WML for wireless browsers, to XHTML, or even to another XML definition for B2B transformations.

XSL Style Sheets

The style sheet itself is a W3C standard. The style sheet is external to the XML document and referenced within the XML document. It must be a well-formed XML document.

The following XSL style sheet, `CHA2_Events.XSL`, formats the sample XML document we have been using thus far, to present a table of events for each tour run by the Cape Hatteras Adventure Company. The style sheet doesn't include any cool graphics or fancy styling so that the focus can be on how the XML data is integrated within the XSL style sheet. However, any good-looking HTML page can be built with XSL style sheets.

The declaration section identifies the style sheet as an XSL document, and references two namespaces. Within `xsl:template match ="/"`, which identifies the root of the style sheet that contains the HTML framework, notice the familiar `<html>` and `<body>` tags. The `<xsl:apply-templates select="Tours"/>` tag is where the Tours template is inserted within the HTML framework.

Within the `<xsl:template match="Tours">` tag, the XML `tour` element is read and the `for-each` element is used much like a loop to iterate through the XML data. The XML's attribute or text data is referenced with the `value-of` command:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="HTTP://www.w3.org/1999/XSL/Transform"
  xmlns:fo="HTTP://www.w3.org/1999/XSL/Format">

  <xsl:template match="/">
    <html>
      <body>
        Cape Hatteras Adventures <br> </br>
        Event Schedule
        <hr></hr>
        <xsl:apply-templates select="Tours"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="Tours">
    <xsl:for-each select= "Tour">
      <xsl:sort select="Name"/>
      <b><xsl:value-of select="@Name"/> </b><br></br>
      <table border="1">
        <xsl:for-each select= "Event">
        <xsl:sort select="DateBegin "/>
          <tr>
            <td><xsl:value-of select="@Code"/></td>
            <td><xsl:value-of select="@DateBegin"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

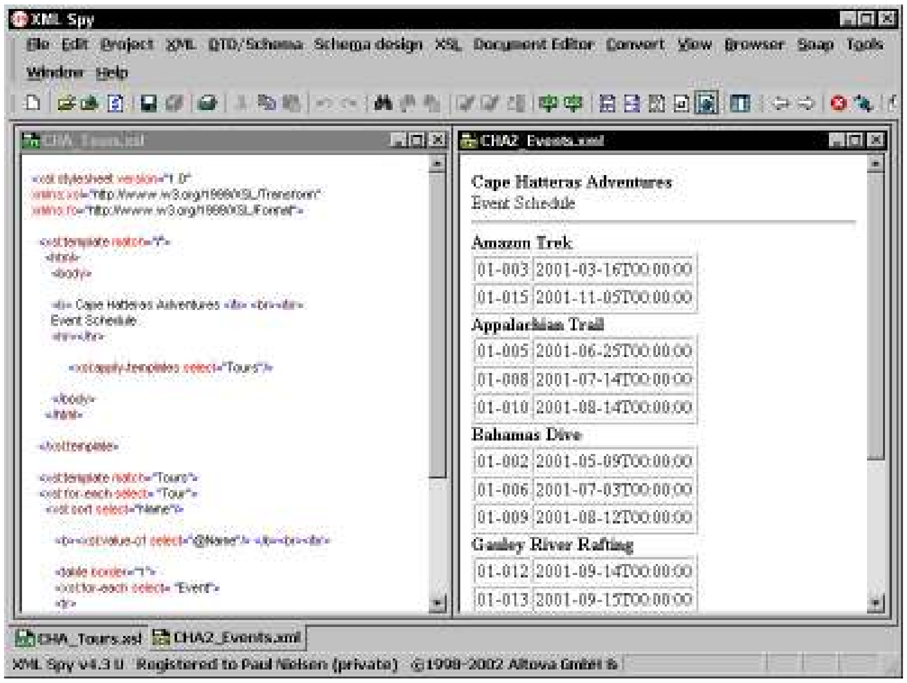
The declaration section of the sample XML document is altered to include a reference to the new style sheet (`CHA2_Events_XSL.xml`):

```
<?xml-stylesheet type="text/xsl" href="CHA_Tours.xsl"?>
```

The XSL style sheet can also ensure the sort order of the data by use of the `xsl:sort` tag:

```
        <xsl:sort select="Name"/>
```

When the XML document is viewed with a browser the result is a basic Web page, as shown in Figure 22-3.


XML Spy
File Edit Project XML DTD/Schema Schema design XSL Document Editor Convert View Browser Soap Tools
Window Help
CHA_Tours.xsl
CHA2_Events.xml
XML Spy v4.3 U Registered to Paul Nielsen (private) ©1998-2002 Altova GmbH &

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">

 <xsl:template match="/">
  <html>
   <body>

   <b> Cape Hatteras Adventures </b> <br></br>
   Event Schedule
   <hr></hr>

      <xsl:apply-templates select="Tours"/>

   </body>
  </html>

 </xsl:template>

 <xsl:template match="Tours">
 <xsl:for-each select="Tour">
  <xsl:sort select="Name"/>

    <b><xsl:value-of select="@Name"/> </b><br></br>

    <table border="1">
    <xsl:for-each select="Event">
    <tr>
```

Cape Hatteras Adventures
Event Schedule

Amazon Trek

01-003	2001-03-16T00:00:00
01-015	2001-11-05T00:00:00

Appalachian Trail

01-005	2001-06-25T00:00:00
01-008	2001-07-14T00:00:00
01-010	2001-08-14T00:00:00

Bahamas Dive

01-002	2001-05-09T00:00:00
01-006	2001-07-03T00:00:00
01-009	2001-08-12T00:00:00

Gauley River Rafting

01-012	2001-09-14T00:00:00
01-013	2001-09-15T00:00:00

insert fg2203.jpg

Figure 22-3: Viewing the XML data with the XSL style sheet applied reads the XML data into an HTML page. While this page isn't very fancy, XSL style sheets can produce any HTML code.

Transforming XML to XML

XSL is simply a means of restyling XML into another format. While it's typically used to transform XML data into an HTML page, with some creative work XSL can transform an XML document into a different XML document, which is very useful to Web developers performing B2B XML transformations.

Combining this technique with XPath queries can "`select`" a subset of data from one XML document into another XML document. The element and attribute structure may also be changed in the transformation. During the creation of the XSL style sheet the `<xsl:element>` and `<xsl:attribute>` tags are used to create the new elements and attributes.

Note
More information on XML transformation and XPath can be found in the "*XML Bible, Second Edition*" (John Wiley & Sons Inc.).

XPATH

XPath is a feature of XML and adds result restriction and aggregation capabilities to XML while navigating through the XML hierarchy to access any node (element, attribute, or piece of text) within the XML document. The XPath code may be added to the XSL style sheet or to other XML queries to filter the data. The most applicable use of XPath for SQL Server is in templates. (While XPath is stable, its more powerful cousin, XQuery, is still evolving.)

XPath must first identify the node. Identifying a node is very similar to the column name within a SQL `select` statement's `where` clause. For example, to navigate to the `event` element under the `tour` element use the following structure:

```
/Tours/Tour/Event
```

The previous example uses absolute location. The node location may also be selected by means of relative location, so that a portion of the XSL that's working with child data may select a parent node for the filtering process.

To navigate down the hierarchical XML tree to an attribute, use the `@` sign, as follows:

```
/Tours/Tour/@Name
```

Once the correct node is identified, XPath has several filtering features to select the correct data. For the execution of a `where =` filter the criteria are specified after the node in square brackets. The file, `CHA2_Events_XPath.xsl`, includes the following XPath code:

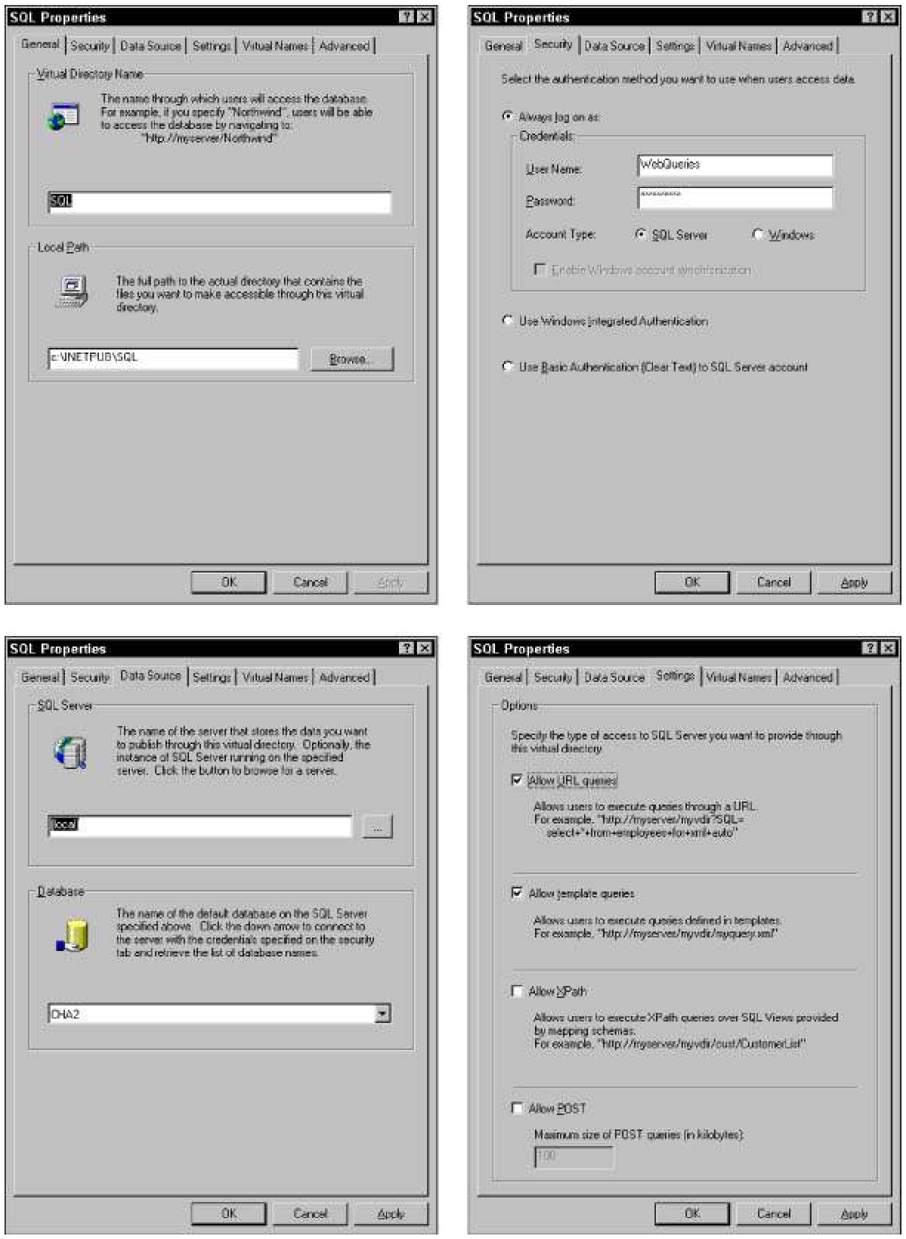
```
/Tours/Tour/@Name ['Gauley River Rafting']
```

SQLXML

Microsoft SQLXML installs as a layer between IIS and SQL Server and allows queries to be submitted to SQL Server from a browser. This is done with virtual directories in IIS.

Virtual Directories

To set up HTTP queries, run the program Configure SQL XML Support in IIS. Right-click the default Web site and select New @@> Virtual Directory. Figure 22-4 shows the dialog box that will appear.


SQL Properties
General | Security | Data Source | Settings | Virtual Names | Advanced
Browse...
OK
Cancel
Apply

Virtual Directory Name
The name through which users will access the database. For example, if you specify "Northwind", users will be able to access the database by navigating to: "http://myserver/Northwind"
SQL

Local Path
The full path to the actual directory that contains the files you want to make accessible through this virtual directory.
c:\INETPUB\SQL

Select the authentication method you want to use when users access data.
(●) Always log on as:
Credentials:
User Name: WebQueries
Password: **********
Account Type: (●) SQL Server ( ) Windows
☐ Enable Windows account synchronization
( ) Use Windows Integrated Authentication
( ) Use Basic Authentication (Clear Text) to SQL Server account

SQL Server
The name of the server that stores the data you want to publish through this virtual directory. Optionally, the instance of SQL Server running on the specified server. Click the button to browse for a server.
local

Database
The name of the default database on the SQL Server specified above. Click the down arrow to connect to the server with the credentials specified on the security tab and retrieve the list of database names.
CHA2

Options
Specify the type of access to SQL Server you want to provide through this virtual directory.
☑ Allow URL queries
Allows users to execute queries through a URL.
For example, "http://myserver/myvdir?SQL=select+*+from+employees+for+xml+auto"
☑ Allow template queries
Allows users to execute queries defined in templates.
For example, "http://myserver/myvdir/myquery.xml"
☐ Allow XPath
Allows users to execute XPath queries over SQL Views provided by mapping schemas.
For example, "http://myserver/myvdir/cust/CustomerList"
☐ Allow POST
Maximum size of POST queries (in kilobytes):
100

Insert fg2204.jpg

Figure 22-4: Configuring Virtual Directories in IIS to run HTTP queries in SQL Server

The General tab of the dialog box will prompt you for a virtual-directory name and local path. Choose a name for the virtual directory that will appear after the server name in the HTTP request. Also, you need to have it point to a physical directory that will not really contain anything, but that needs to exist in order for the configuration to work. Choose a simple name like SQL or the name of the database against which the queries will run.

On the Security tab, enter the login ID the queries will be run as. The best approach is to create an account for the queries to run under. You can specify either a SQL or Windows login for this. Another approach is to require the browser to pass a client's Windows authentication if the session is taking place within an intranet; alternately, you can send a text login ID and password that would typically be stored in a database or configuration file. It is somewhat dangerous to pass clear-text login ID's and passwords and this is not a recommended practice.

The Data Source tab is used to specify the server name and default directory for the queries. The Settings tab is where you enable the features you want to support. Allowing URL queries enables people to execute ad hoc queries against the SQL Server. Allowing template queries enables clients to only execute queries that have been previously saved into a template file. URL queries can be dangerous if you open up too much access to the account the queries run under, whereas template queries limit the clients to only those queries that have been preprogrammed.

Allowing XPath queries requires an XDR schema file to be created for each query that maps the columns to XML attributes. The URL query can then execute the SQL statement and use the XPath language to drill down into a particular node.

HTTP Queries

If the virtual directory is enabled and allows HTTP queries, a browser pointed to the correct server can issue a query in the address line of the browser using the `for xml` syntax:

```
HTTP://server/virtualdirectory?sql=SELECT QUERY&
  Root=RootName
```

The select query cannot include any spaces, so substitute a + for any spaces. Also, the `select…for xml` command does not generate a well-formed XML document because it lacks an XML declaration section and root node. The second parameter, `root`, supplies the name of the root node. For example, the following HTTP query retrieves a list of tours and start dates into an XML document with a root named Tours:

```
HTTP://localhost/sql?sql=SELECT+Tour.Name,+Event.Code,+Event.DateBegin+FROM+Tour
+JOIN+Event+ON+Tour.TourID+=+Event.TourID+FOR+XML+AUTO,+ELEMENTS&root=Tours
```

The result of the HTTP query is shown in Figure 22-5.

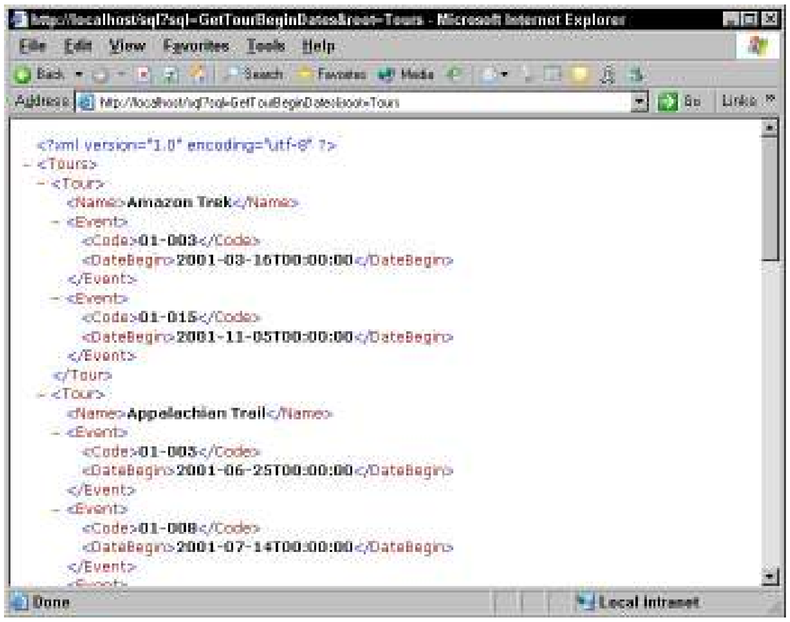
Note that you may type the query with spaces, or with plus signs instead of spaces. However, if you use spaces, the Web server will convert each one to `%20`, which makes the URL difficult to read. The better way to handle this and make the URL more readable is to save the whole query in a stored procedure:

```
CREATE PROCEDURE GetTourBeginDates AS
SELECT Tour.Name, Event.Code, Event.DateBegin
FROM Tour JOIN Event ON Tour.TourID = Event.TourID
FOR XML AUTO, ELEMENTS
```

Then simply use the stored-procedure name in the HTTP query:

```
HTTP://localhost/sql?sql=GetTourBeginDates&root=root
```

The results will be identical but it will be much easier to manage and view the URL.


http://localhost/sql?sql=GetTourBeginDates&root=Tours - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/sql?sql=GetTourBeginDates&root=Tours
Go Links
Done
Local intranet

```
  <?xml version="1.0" encoding="utf-8" ?>
- <Tours>
  - <Tour>
      <Name>Amazon Trek</Name>
    - <Event>
        <Code>01-003</Code>
        <DateBegin>2001-03-16T00:00:00</DateBegin>
      </Event>
    - <Event>
        <Code>01-015</Code>
        <DateBegin>2001-11-05T00:00:00</DateBegin>
      </Event>
    </Tour>
  - <Tour>
      <Name>Appalachian Trail</Name>
    - <Event>
        <Code>01-005</Code>
        <DateBegin>2001-06-25T00:00:00</DateBegin>
      </Event>
    - <Event>
        <Code>01-008</Code>
        <DateBegin>2001-07-14T00:00:00</DateBegin>
      </Event>
```

insert fg2205.jpg

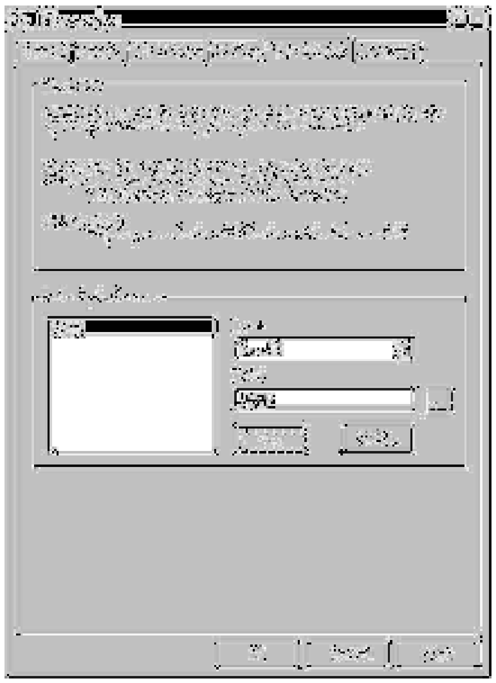
Figure 22-5: Viewing the XML data, with the XSL style sheet applied, reads the HTTP queries.

Template Queries

Template queries enable you to assert a little more control over what queries can be run over an HTTP request, as well as over how the output will look. First you need to configure the virtual directory to allow template queries. Then create a Virtual Name as a template type and a directory to contain the template files. Once that is done, you need to create a template file and save it in the directory specified in the Virtual Name configuration. You can then run queries using the following format:

```
HTTP://server/virtualdirectory/virtualname/filename.xml
```

The XML template file is a fully formed XML document, but where you want the contents of a query substituted into the document, place the SQL statement between the tags `<sql:query>` and `</sql:query>`. Configure a virtual name called Temp with a type of template that points to a directory called `\temp` (see Figure 22-6). Make sure you create a subdirectory under the physical directory that the virtual directory points to.



insert fg2206.jpg

Figure 22-6: Allow template queries via the Virtual Names tab.

Once the virtual name exists, create a document called `Tours.xml` and save it in the path `C:\INETPUB\SQL\Temp\Tours.XML`.

```
<?xml version ='1.0' encoding='UTF-8'?>
  <root xmlns:sql="urn:schemas-microsoft-com:xml-sql">
    <sql:query>
         SELECT Tour.Name, Event.Code,
   Event.DateBegin
         FROM Tour
         JOIN Event ON Tour.TourID = Event.TourID
         FOR XML AUTO, ELEMENTS
    </sql:query>
  </root>
```

To execute this template query all you need to do is run the following:

```
HTTP://localhost/sql/Temp/Tours.xml
```

Running this will yield the same result as running the URL query, but it has been set up in advance so it does not open the door to potentially dangerous ad hoc queries. It also simplifies the URL for queries that are run frequently.

Publishing Data on the Web

Apart from generating XML, SQL Server has the ability to generate HTML directly from a stored procedure.

The Web Assistant is a wizard in Enterprise Manager that can be used to generate calls to stored procedures that will generate HTML documents from a query. The documents it creates are called *static HTML files*, because a query is run at a given time and it generates a result set that is converted into HTML and saved as a file. The file itself is returned to the client. This method is different from a dynamic HTML page, which is generated through a script file such as ASP, ColdFusion, or JSP. With a dynamic page, a query is run against the database for each call to the Web page and a new HTML document is generated in memory for each client request. Both dynamic and static pages have advantages and disadvantages. For data that does not change much, such as a phone list or monthly-summary data, a static page is better because it eliminates the need for excessive calls to the SQL Server. For data that changes more frequently, such as an order-status or user-feedback page, the capabilities of a dynamic page are more appropriate.

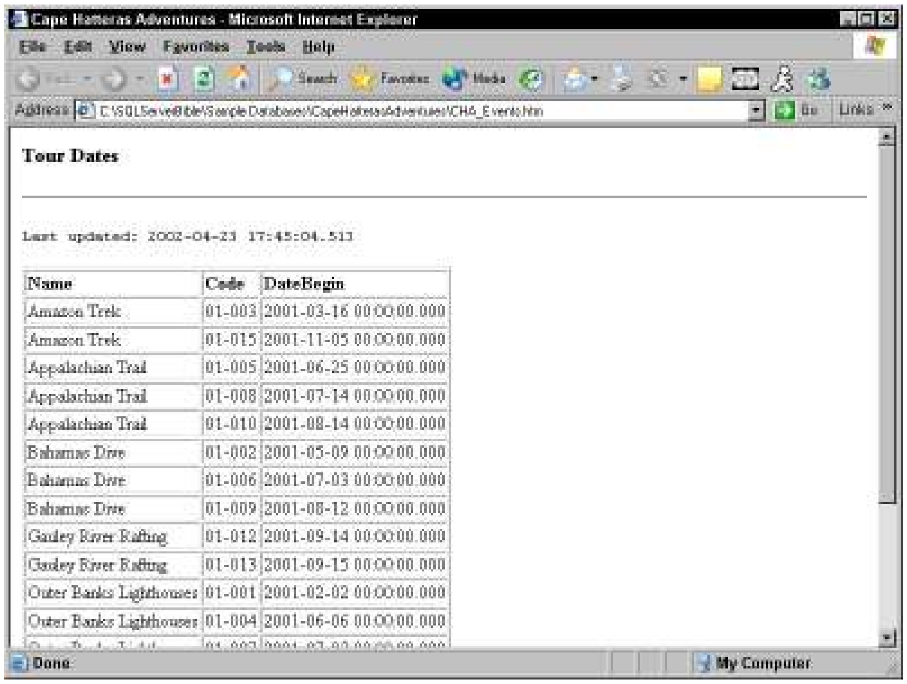
The wizard is relatively easy to use, and you have only a few things to decide on when running it. You can allow the wizard to do basic formatting for you, or you can build a template file using any HTML-authoring tool. The template file enables you to make a page that fits seamlessly into the overall look and feel of the Web site. All that is necessary to make the template file work is the inclusion of a few special tags in the HTML files. If you want to dump the result set of a query into the HTML document as a simple HTML table, just use the tag `<%insert_data_here%>` where you want the table to appear. To have a little more control over the look of the output, use `<%begindetail%>` and `<%enddetail%>` tags, and between them type HTML code that you want repeated for each row of output. Use the tag `<%insert_data_here%>` for each column of the result set to control where they appear in the HTML code.

The other key decision to make is how frequently the Web page will be regenerated. Regeneration can be scheduled as a regular task, or it can occur each time the data changes. Regeneration is no different from any other scheduled job, it simply runs the stored procedure `sp_runwebtask`. To make the Web page refresh each time any data changes, you can create triggers on the table that is to be monitored for changes. Then, whenever there is an `insert`, `update`, or `delete` to that table, it will fire the trigger that runs the same `sp_runwebtask` stored procedure that you would use to recreate the page on a schedule. The advantage of triggers is that the pages are recreated immediately after the change is committed. The disadvantage is that if numerous changes are necessary, the trigger may fire after each change, causing the page to be recreated numerous times leading to serious performance drag.

Using the Web assistant does all the hard work of assembling a call to a stored procedure with a lot of parameters. Here is a sample of the `sp_makewebtask` procedure to create a simple Web page:

```
EXECUTE sp_makewebtask
  @outputfile = N'C:\SQLServerBible\Sample Databases
      \CapeHatterasAdventures\CHA_Events.htm',
  @query=N'SELECT Tour.Name, Event.Code, Event.DateBegin
             FROM Tour
               JOIN Event
                 ON Tour.TourID = Event.TourID',
  @fixedfont=0,
  @HTMLheader=3,
  @webpagetitle=N'Cape Hatteras Adventures',
  @resultstitle=N'Tour Dates',
  @URL=N'HTTP://www.SQLServerBible.com',
  @reftext=N'www.SQLServerBible.com',
  @dbname=N'CHA2',
  @whentype=1,
  @procname=N'CHA2 Web Page',
  @codepage=65001,@charset=N'utf-8'
```

Running the stored procedure `sp_runwebtask @procname = N'CHA2 Web Page'` against the CHA database generates an HTML page (`CHA_Events.htm`) as seen below in Figure 22-7.


Cape Hatteras Adventures - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Search Favorites Media
Address C:\SQLServerBible\Sample Databases\CapeHatterasAdventures\CHA_Events.htm
Go Links
Done
My Computer

Tour Dates

```
Last updated: 2002-04-23 17:45:04.513
```

Name	Code	DateBegin
Amazon Trek	01-003	2001-03-16 00:00:00.000
Amazon Trek	01-015	2001-11-05 00:00:00.000
Appalachian Trail	01-005	2001-06-25 00:00:00.000
Appalachian Trail	01-008	2001-07-14 00:00:00.000
Appalachian Trail	01-010	2001-08-14 00:00:00.000
Bahamas Dive	01-002	2001-05-09 00:00:00.000
Bahamas Dive	01-006	2001-07-03 00:00:00.000
Bahamas Dive	01-009	2001-08-12 00:00:00.000
Gauley River Rafting	01-012	2001-09-14 00:00:00.000
Gauley River Rafting	01-013	2001-09-15 00:00:00.000
Outer Banks Lighthouses	01-001	2001-02-02 00:00:00.000
Outer Banks Lighthouses	01-004	2001-06-06 00:00:00.000

insert fg2207.jpg

Figure 22-7: The Web assistant generates an HTML page of data with our query results.

Summary

This chapter has briefly highlighted how SQL Server can serve data to the Web. We have explored the use of DTDs to validate XML data as well as how to properly implement them. We then touched on the use of style sheets to format XML data for presentation within a Web browser.

This chapter demonstrates the power SQL Server introduces with the use of XML. Unfortunately, due to space limitations, the chapter just scratches the surface of the vast use of XML. To learn more about XML, I highly suggest picking up the *XML Bible* (John Wiley & Sons, Inc.).