

Leer y escribir datos XML

Índice

| | |
|--|----|
| Descripción | 1 |
| Lección: Descripción de la arquitectura XML en ASP.NET | 2 |
| Lección: XML y el objeto DataSet | 10 |
| Lección: trabajar con datos XML | 25 |
| Lección: uso del control de servidor Web XML | 35 |

Descripción

- Descripción de la arquitectura XML en ASP.NET
- XML y el objeto DataSet
- Trabajar con datos XML
- Utilizar el control de servidor Web XML

Introducción

Aunque se almacena una gran cantidad de datos en bases de datos Microsoft® SQL Server™ y se gestionan mediante Microsoft ADO.NET, recientemente Extensible Markup Language (XML) se ha convertido en un sólido estándar para el almacenamiento, gestión y transferencia de datos. XML tiene dos grandes ventajas en el almacenamiento y transferencia de datos:

- XML es un estándar aceptado por el mercado.
- XML utiliza únicamente texto plano.

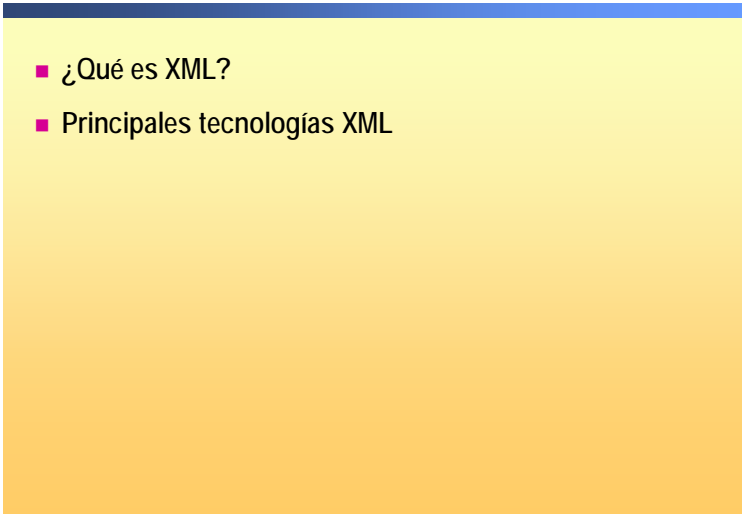
En este módulo, estudiaremos cómo leer, escribir y mostrar datos XML.

Objetivos

En este módulo, aprenderemos a:

- Describir la arquitectura XML en Microsoft ASP.NET.
- Leer y escribir datos XML en un objeto **DataSet**.
- Identificar cómo almacenar, recuperar y transformar datos XML utilizando los objetos **XmlDataDocument** y **XslTransform**.
- Utilizar el control de servidor Web XML para mostrar, cargar y almacenar datos XML.

Lección: descripción de la arquitectura XML en ASP.NET

- 
- ¿Qué es XML?
 - Principales tecnologías XML

Introducción

ASP.NET proporciona varios tipos de clases y objetos que pueden utilizarse para acceder y sincronizar con datos XML. Estas clases y objetos representan componentes para procesar XML a bajo nivel que permiten integrar XML en las aplicaciones Web ASP.NET.

En esta lección, estudiaremos cómo está compuesto un documento XML. También estudiaremos las principales tecnologías XML. Estudiaremos las clases y objetos que se utilizan para leer y escribir datos XML.

Objetivos de la lección

En esta lección, aprenderemos a:

- Distinguir entre XML válido y XML con un formato correcto.
- Describir las principales tecnologías XML.

¿Qué es XML?

El diagrama muestra un fragmento de código XML con las siguientes partes anotadas:

- Instrucción de procesamiento:** `<?xml version="1.0"?>`
- Elementos:** `<authors>` y `<author ID="1">`
- Atributos:** `ID="1"`
- Comentarios:** `<!-- There are more authors. -->`

El código XML completo mostrado es:

```
<?xml version="1.0"?>
<authors>
  <author ID="1">
    <name>Jay</name>
  </author>
  <!-- There are more authors. -->
</authors>
```

Además, se listan las siguientes características de XML:

- Proporciona un método uniforme para describir e intercambiar datos estructurados
- Podemos definir nuestros propios elementos y atributos
- Los elementos pueden anidarse
- XML válido frente a XML con un formato correcto

Introducción

Actualmente, las empresas se enfrentan a numerosos problemas en la organización de los datos porque necesitan cumplir con los siguientes requerimientos:

- Las estaciones de trabajo y los usuarios necesitan leer datos.
- Tanto el contenido como la estructura de los datos deben estar definidos.
- La estructura de los datos debe estar separada de su presentación.
- La estructura de los datos debe ser abierta y extensible.

XML satisface todos estos requerimientos, y por tanto es una ayuda para la organización de datos en las empresas.

Definición

XML es el formato universal utilizado para describir e intercambiar documentos y datos estructurados en Internet. XML es un subconjunto del Standard Generalized Markup Language (SGML), y está definido por el World Wide Web Consortium (W3C), y garantiza que los datos estructurados son uniformes e independientes de las aplicaciones Web y de los fabricantes.

XML define la estructura de los datos de un modo abierto y autodestructivo. Este modo abierto y autodestructivo permite que los datos sean fácilmente transferidos a través de una red y que el receptor los procese de modo coherente. XML describe cómo se estructuran los datos, no cómo deberían mostrarse o utilizarse, de forma similar a Hypertext Markup Language (HTML). Los documentos XML contienen etiquetas que otorgan significado al contenido del documento. Estas etiquetas permiten a los programadores encontrar los datos que necesitan en el documento XML.

Partes de un documento XML

Las partes de un documento XML incluyen:

- Instrucciones de procesamiento
- Elementos
- Atributos
- Comentarios

| | |
|------------------------------|--|
| Instrucción de procesamiento | La mayoría de documentos XML empiezan con una instrucción de procesamiento para el procesador XML indicando que el documento sigue las recomendaciones XML del W3C. |
| Elementos | <p>A continuación de la instrucción de procesamiento puede definirse un conjunto de elementos anidados. Respecto a los elementos:</p> <ul style="list-style-type: none">■ Normalmente, un elemento está formado por un par de etiquetas: una etiqueta de inicio y una etiqueta de cierre.■ Entre el par de etiquetas de inicio y cierre, un elemento puede contener el contenido de datos u otros elementos.■ Un elemento puede estar formado únicamente por la etiqueta de cierre.■ El primer elemento que encuentra el procesador XML debe estar formado por una etiqueta de inicio y una etiqueta de cierre. El primer elemento contiene el resto de elementos y se denomina elemento raíz.■ El resto de elementos, tras el primero, pero dentro del elemento raíz, se denominan elementos secundarios.■ Los elementos secundarios pueden anidar otros elementos secundarios. La mayoría de los datos del contenido XML se almacenan entre la etiqueta de inicio y la etiqueta de cierre de los elementos secundarios. |
| Atributos | <p>Cualquier elemento puede contener atributos. El uso de atributos es una alternativa al uso de elementos para almacenar el contenido. Los atributos definen datos que pertenecen a un único elemento. Respecto a los atributos:</p> <ul style="list-style-type: none">■ Crear un atributo en la etiqueta de inicio de un elemento.■ Declarar el nombre del atributo, seguido de una asignación del valor.■ Utilizar comillas simples o dobles para establecer el valor de un atributo. |
| Comentarios | Los comentarios son opcionales. |
| XML de formato correcto | <p>Un documento XML con un formato correcto cumple las especificaciones listadas en la Recomendación W3C de XML 1.0. Un documento XML se considera con el formato correcto si:</p> <ul style="list-style-type: none">■ Contiene exactamente un elemento raíz (el elemento documento).■ Todos los elementos secundarios están anidados correctamente uno dentro de otro.■ Existen las etiquetas de inicio y final de un determinado elemento en el cuerpo del mismo elemento primario. <p>Ejemplo de XML con el formato correcto:</p> <pre><Temp>22</Temp></pre> <p>Ejemplo de XML mal formado:</p> <pre><Temp>22</temp></pre> <p>El error en el ejemplo anterior es que la etiqueta de cierre <temp> no coincide con la etiqueta de inicio <Temp>.</p> |

XML válido

El XML es válido si su vocabulario cumple con una serie de requisitos descritos en un esquema:

- En XML, un esquema es la descripción de un documento XML.
- Un esquema se utiliza para validar documentos XML. Los documentos XML validados con un esquema se denominan documentos instancia. Si un documento instancia coincide con la definición del esquema, el documento instancia se considera válido.

Existen tres tipos de esquemas que pueden utilizarse para validar un documento instancia XML, como muestra la siguiente tabla.

| Tipo de esquema | Descripción |
|--------------------------------------|--|
| Document Type Definition (DTD) | DTD es el método de validación original descrito en la Recomendación XML versión 1.0 del W3C. XML Schema Definition (XSD) ha sustituido a DTD. Los DTDs no están basados en XML. |
| XML-Data Reduced (XDR schema) | XDR es una tecnología de esquema provisional desarrollada por Microsoft. XDR es similar a XSD, pero los esquemas XDR están escritos en XML. |
| XML Schema Definition language (XSD) | XSD es la recomendación del W3C para validar esquemas XML. XSD sustituye a ambos esquemas, DTDs y XDR. Los esquemas XSD están escritos en XML. |

XSD es el esquema utilizado más frecuentemente en el .NET Framework.

Principales tecnologías XML

- Definición de esquemas XML (XSD)
 - Define la estructura requerida de un documento XML válido
- Extensible Stylesheet Language Transformation (XSLT)
 - Transforma el contenido de un documento XML fuente en otro documento que es diferente en formato o estructura
- Lenguaje XML Path (XPath)
 - Direcciona partes de un documento XML
- Document Object Model (DOM)
 - Modelo de objetos para trabajar programáticamente con documentos XML en memoria
- XML Query (XQuery)
 - Lenguaje de fácil implementación en el que las consultas son concisas y se entienden fácilmente

Introducción

Desde el principio, XML ha generado otras innovaciones tecnológicas y desarrollos que funcionan con XML para manipular datos. Las principales tecnologías relacionadas con XML, todas ellas recomendaciones del W3C, incluyen:

- Definición de esquemas XML (*XSD*)
- Extensible Stylesheet Language Transformation (*XSLT*)
- Lenguaje XML Path (*XPath*)
- Document Object Model (*DOM*)
- Consulta XML (*XQuery*)

XSD

XSD es el estándar actual para la definición de esquemas, y define la estructura que requiere un documento XML válido. Podemos crear un esquema XSD como un documento aislado y que sea referenciado por documentos instancia. Un documento instancia es un documento XML validado por un esquema XML. También podemos incluir un esquema XSD en un documento XML. La extensión de un archivo de esquema aislado es .xsd.

Definición de esquema

Podemos definir un documento XML como un esquema utilizando el elemento `<xsd:schema>`. El espacio de nombres esquema del W3C cualifica el prefijo `xsd:`. Cada elemento que identificamos con el prefijo `xsd:` pertenece al espacio de nombres XSD.

En XSD, podemos referenciar múltiples espacios de nombres (`xmlns`). Por ejemplo, la siguiente definición de esquema referencia a dos espacios de nombres, el primero para el XML Schema del W3C, y el segundo para un esquema de datos de Microsoft Office 10:

```
<xsd:schema xmlns:xsd="http://www.w3c.org/2000/10/XMLSchema"
xmlns:od="urn:schemas-microsoft-com:officedata">
```

Declaraciones de elemento y atributo

Utilizamos declaraciones de elemento y atributo para definir el uso de los elementos y los atributos en un documento XML. Podemos definir las siguientes propiedades para un elemento o atributo: nombre, contenido, número, secuencia de ocurrencias, tipo de datos.

En el siguiente ejemplo, el nombre del elemento se declara como <LastName>. En el documento, <LastName> puede ocurrir 0 o más veces. El tipo de elemento es una cadena:

```
<xsd:element name="LastName" minOccurs="0" maxOccurs="*" type="string"></xsd:element>
```

XSLT

Por muchas razones, los datos XML deben transformarse en otras formas y variantes. El W3C ha generado XSLT como uno de los lenguajes de programación que puede ser utilizado para transformar datos. XSLT es una parte de XSL (*eXtensible Stylesheet Language*).

XSLT es un lenguaje basado en XML que transforma documentos XML a formatos arbitrarios basados en texto, que pueden ser o no XML.

Los tres documentos siguientes se utilizan con XSLT:

- El documento origen

El documento origen es simplemente un documento XML con un formato correcto que proporciona la entrada para la transformación. Por ejemplo, el siguiente código es una muestra de un documento origen XML:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="Employees1.xsl"?>
<employees>
  <employee>
    <name>Stuart Munson</name>
    <jobtitle>Programmer</jobtitle>
  </employee>
  <employee>
    <name>Robert Brown</name>
    <jobtitle>Tester</jobtitle>
  </employee>
</employees>
```


- Documento de hoja de estilo XSLT

El documento de hoja de estilo XSLT es un documento XML que utiliza el vocabulario de XSLT para expresar las normas de transformación. Por ejemplo, el siguiente código es el documento de hoja de estilo XSLT (Employees1.xsl) que se aplicará al documento origen del código anterior:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
    <xsl:apply-templates select="//employee" />
  </xsl:template>
  <xsl:template match="employee">
    <P>
      <xsl:apply-templates />
      <HR />
    </P>
  </xsl:template>
  <xsl:template match="name">
    <FONT COLOR="red" />
    <B>
      <xsl:value-of select="." />
    </B>
  </xsl:template>
  <xsl:template match="jobtitle">
    <BR/>
    <FONT COLOR="blue" />
    <xsl:value-of select="." />
  </xsl:template>
</xsl:stylesheet>
```

- El documento resultante

El documento resultante es un documento de texto producido por la ejecución del documento origen a través de las transformaciones que se encuentran en la hoja de estilo XSLT. Por ejemplo, ejecutando el documento origen del código anterior a través de la hoja de estilo XSLT anterior Employees1.xsl, se produce el siguiente documento resultante:

Stuart Munson
Programmer

Robert Brown
Tester

| | |
|--------|--|
| XPath | <p>XPath es un lenguaje completo utilizado para referenciar elementos en los documentos XML. XPath versión 2.0 es una recomendación del W3C.</p> <p>El lenguaje XPath especifica un modelo de objetos para los documentos XML. En el modelo de objetos XPath, un documento XML está representado como un árbol de nodos. Consultamos una fuente XML utilizando las propiedades de sus nodos XPath.</p> |
| DOM | <p>DOM es una representación en forma de árbol en el caché de memoria de un documento XML. DOM permite la navegación y edición de un documento XML. W3C define las propiedades, métodos y eventos de DOM.</p> <p>Con ASP.NET, podemos escribir un script que se ejecuta en el servidor Web y que utiliza el DOM para crear un documento XML que se enviará al navegador. También podemos escribir un script del lado del cliente que genere un documento XML en el cliente envíe los datos XML al servidor Web, cuando resulte apropiado.</p> |
| XQuery | <p>A medida que va aumentando la cantidad de información que se almacena, intercambia y presenta mediante XML, la capacidad de consultar inteligentemente las fuentes de datos XML es cada vez más importante. XQuery proporciona características para recuperar e interpretar información desde estas fuentes de datos:</p> <ul style="list-style-type: none">■ XQuery ha sido diseñado como un lenguaje de fácil implementación en el que las consultas son concisas y se entienden fácilmente.■ La flexibilidad de XQuery permite consultar un amplio espectro de fuentes de información XML, incluyendo tanto bases de datos como documentos.■ XQuery se basa en otras tecnologías del W3C. Por ejemplo, XQuery utiliza sentencias de ruta de la recomendación XPath 2.0.■ XQuery depende en gran medida de XPath para dirigir sus consultas dentro de un determinado origen XML. XQuery también toma prestado el modelo de objetos de XPath. |

Nota Para más información sobre las principales tecnologías XML, acceder al sitio Web del W3C en <http://www.w3c.org>.

Lección: XML y el objeto DataSet

- ¿Por qué utilizar XML con DataSets?
- Descripción de XML y DataSets
- Métodos basados en XML del objeto DataSet
- Demostración: leer y escribir XML a/desde un DataSet
- Práctica: uso del método ReadXml
- Crear datos XML anidados
- Demostración: crear XML anidado

Introducción

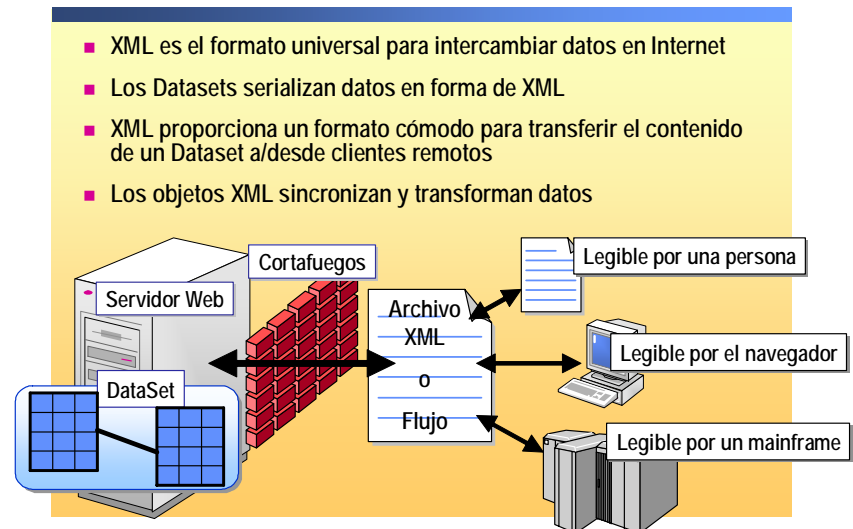
XML y los **DataSets** comparten una estrecha conexión. Los **DataSets** son la base para el almacenamiento en modo desconectado y la manipulación de datos relacionales. Los **DataSets** también son un contenedor para una o más tablas de datos. XML es el formato estándar para los datos presentes en **DataSets**.

Objetivos de la lección

En esta lección, aprenderemos a:

- Describir el uso de XML con DataSets.
- Identificar la relación de XML con DataSets.
- Identificar el uso de los métodos **ReadXml**, **WriteXml** y **GetXml**.
- Crear datos XML anidados.

¿Por qué utilizar XML con DataSets?



Introducción

XML es un formato universal utilizado para intercambiar datos en Internet, y un **DataSet** es una vista relacional de datos que puede representarse en XML. XML se utiliza con los DataSets de las siguientes formas:

- Serializar datos

Los DataSets pueden serializar datos en forma de XML. El esquema de un DataSet que incluye tablas, columnas, tipos de datos y restricciones se define utilizando un esquema XML (archivo .xsd).

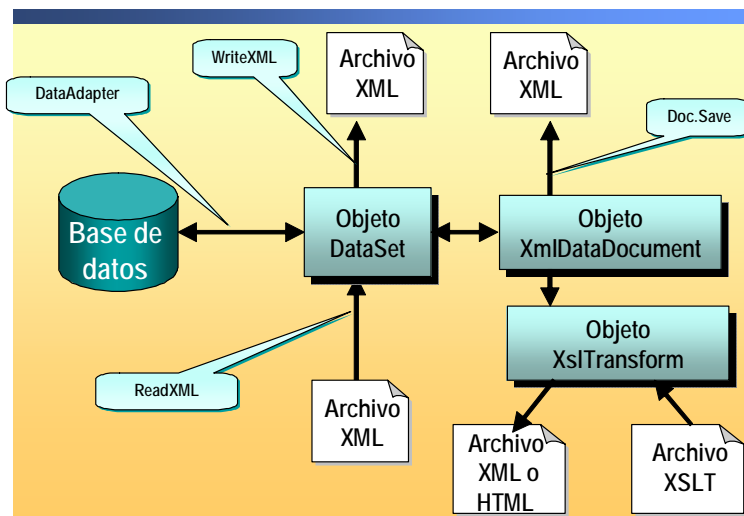
- XML y XML Schema

XML y los esquemas XML Schema proporcionan un formato cómodo para transferir el contenido de un DataSet a/desde clientes remotos. Podemos inferir esquemas XML de DataSets existentes y crear DataSets desde esquemas XML existentes.

- Sincronizar y transformar datos

Podemos utilizar diferentes objetos XML para sincronizar y transformar datos que están representados por DataSets.

Descripción de XML y DataSets



Introducción

XML desempeña un importante papel en el modo en que el .NET Framework gestiona los datos. XML es el formato utilizado en el .NET Framework para almacenar y transferir todo tipo de datos. Los **DataSets** pueden almacenar y transferir datos en formato XML. Respecto a las características de los **DataSets** y XML:

- La estructura de un DataSet puede definirse en un esquema XML Schema

La estructura de un DataSet que incluye tablas, columnas, relaciones y restricciones puede definirse utilizando un esquema XML. Los esquemas XML son un formato basado en estándares del W3C que pueden utilizarse para definir la estructura de datos XML.

- Generar una clase DataSet

Podemos generar una clase DataSet que incorpore información de esquema para definir sus estructuras de sus datos (como tablas y columnas) como miembros de clase.

- Métodos DataSet

Podemos leer un documento XML o generar un flujo en un DataSet utilizando el método **ReadXML** del DataSet y a continuación escribir un DataSet en XML utilizando el método **WriteXML** del DataSet. Como XML es un formato estándar para intercambiar datos entre distintas aplicaciones Web, podemos cargar un DataSet con información formateada en XML que haya sido enviada por otras aplicaciones. De forma similar, un DataSet puede escribir sus datos como un flujo XML o un documento que será compartido con otras aplicaciones o simplemente almacenado como un documento XML.

- Crear una vista XML del contenido de un DataSet

Podemos crear una vista XML (un objeto **XmlDataDocument**) del contenido de un DataSet, y visualizar y manipular los datos utilizando métodos relacionales (mediante el DataSet) o métodos XML. Las dos vistas se sincronizan automáticamente cuando se modifican.

- Transformación de datos

Podemos utilizar el objeto **XSLTransform** para cargar una hoja de estilo .xsl y aplicar la transformación. El documento resultante puede ser un archivo XML o HTML.

Nota Los DataSets pueden leer y escribir esquemas que almacenan información estructurada utilizando los métodos **ReadXmlSchema** y **WriteXmlSchema**. Si no hay ningún esquema disponible, el DataSet puede producir uno, mediante su método **InferXmlSchema**, a partir de los datos existentes en un documento XML que esté estructurado en un modo relacional.

Los métodos basados en XML del objeto DataSet

■ Utilizan ReadXml para cargar datos de un archivo o flujo

```
DataSet ds = new DataSet();
ds.ReadXml(Server.MapPath("filename.xml"));
```

■ Utilizan WriteXml para escribir datos XML a un archivo o flujo

```
DataSet ds = new DataSet();
SqlDataAdapter da = new SqlDataAdapter("select * from
    Authors", conn);
da.Fill(ds);
ds.WriteXml(Server.MapPath("filename.xml"));
```

■ Utilizan GetXml para escribir datos a una variable de cadena

```
string strXmlDS = ds.GetXml();
```

[Código de ejemplo de Visual Basic .NET](#)

Introducción

El contenido de un **DataSet** puede crearse desde un flujo o un documento XML. Además, con el .NET Framework, tenemos una gran flexibilidad sobre qué información se carga desde XML, y cómo se crea el esquema o estructura relacional del **DataSet**.

ReadXML

Para rellenar un **DataSet** con datos de XML, utilizamos el método **ReadXml** del objeto **DataSet**. El método **ReadXml** lee de un archivo, un flujo o un **XmlReader**.

El método **ReadXml** lee el contenido del flujo o documento XML y carga el **DataSet** con esos datos. **ReadXml** también crea el esquema relacional del **DataSet**, dependiendo del **XmlReadMode** especificado y de si existe ya o no un esquema relacional.

El siguiente código muestra cómo rellenar un DataSet con datos:

Microsoft Visual Basic® .NET

```
Dim ds As New DataSet()
ds.ReadXml(Server.MapPath("filename.xml"))
```

C#

```
DataSet ds = new DataSet();
ds.ReadXml(Server.MapPath("filename.xml"));
```

Nota El método **Server.MapPath** devuelve la ruta de archivo física que corresponde a la ruta virtual especificada en el servidor Web.

WriteXML

Para escribir un **DataSet** a un archivo, flujo o **XmlWriter**, utilizamos el método **WriteXml**. El primer parámetro que pasamos a **WriteXml** es el destino de la salida XML. Por ejemplo, podemos pasar una cadena que contenga un nombre de archivo, un objeto **System.IO.TextWriter**, etc. Podemos pasar un segundo parámetro opcional de un **XmlWriteMode** para especificar cómo debe escribirse la salida XML.

El siguiente código es un ejemplo de cómo escribir un DataSet:

Visual Basic .NET

```
Dim ds As New DataSet()  
Dim da As New SqlDataAdapter( _  
    "select * from Authors", conn)  
da.Fill(ds)  
ds.WriteXml(Server.MapPath("filename.xml"))
```

C#

```
DataSet ds = new DataSet();  
SqlDataAdapter da = new SqlDataAdapter  
    ("select * from Authors", conn);  
da.Fill(ds);  
ds.WriteXml(Server.MapPath("filename.xml"));
```

GetXML

La representación XML del **DataSet** puede escribirse en un archivo, un flujo, un **XmlWriter** o una cadena. Estas opciones proporcionan una gran flexibilidad para el modo de transporte de la representación XML del **DataSet**. Para obtener la representación XML del **DataSet** como una cadena, utilizaríamos el método **GetXml**, como muestran los siguientes ejemplos de código:

Visual Basic .NET

```
Dim strXmlDS As String = ds.GetXml()
```

C#

```
string strXmlDS = ds.GetXml();
```

GetXml devuelve la representación XML del **DataSet** sin información de esquema. Para escribir la información de esquema desde el **DataSet** (como el esquema XML) a una cadena, utilizamos **GetXmlSchema**.

Demostración: leer y escribir XML a y desde un DataSet



Leer XML

- Crear un DataSet
- Cargar un DataSet desde un archivo XML
- Mostrar en un DataGrid

Escribir XML

- Crear un DataSet desde la base de datos
- Crear un archivo XML desde un DataSet

Introducción

En esta demostración, veremos cómo leer y escribir XML a/desde un **DataSet**.

Los archivos de esta demostración se encuentran en los proyectos Demo10CS y Demo10VB que se pueden encontrar dentro del fichero demos10.zip

🔗 Ejecutar la demostración

Leer datos XML

1. Abrir el archivo Books.xml en Microsoft Visual Studio® .NET.
Este archivo contiene los datos que se mostrarán.
2. Generar y examinar la página DisplayXML.aspx.
3. En el cuadro de texto, hacer clic en **Books.xml** y clic en **Load**.
4. En el cuadro de texto, hacer clic en **Employees.xml** y clic en **Load**.
5. Abrir el archivo de código subyacente DisplayXML.aspx.vb o DisplayXML.aspx.cs en Visual Studio .NET.

6. En el procedimiento de evento **cmdLoad_Click**, mostrar el código que lee un archivo XML en un **DataSet** y vincula el **DataGrid** al **DataSet**.

Nota El **DataGrid** únicamente puede gestionar un único nivel de elementos en un archivo XML.


Si hay demasiada anidación de elementos, los datos no se mostrarán. Podemos demostrar una anidación excesiva agregando un elemento *author* a los elementos *book* en el archivo Books.xml:

```
<book>
...
  <author>
    <firstname>Jay</firstname>
    <lastname>Bird</lastname>
  </author>
</book>
```

Escribir datos XML

7. Generar y examinar la página DisplayXML.aspx.
No se muestran los datos debido a la excesiva anidación.
8. Generar y examinar la página SaveAsXML.aspx.
El **DataGrid** muestra los datos del **DataSet** que se guardarán en un archivo XML.
9. Hacer clic en **Save as XML**, y clic en el hipervínculo **View XML**.
Estos son los datos XML que se han creado desde el **DataSet**.
10. Abrir uno de los archivos de código subyacente SaveAsXml.aspx.vb o SaveAsXml.aspx.cs en Visual Studio .NET.
Hay una función denominada **CreateDataSet** que genera el **DataSet** desde una base de datos SQL Server.
11. Para crear un archivo XML, mostrar el código en el procedimiento de evento **cmdSave_Click** que invoca el método **WriteXml** del **DataSet**.
12. Para crear un archivo de esquema XSD, mostrar el código en el procedimiento de evento **cmdSchema_Click** que invoca el método **WriteXmlSchema** del **DataSet**.

Práctica: uso del método ReadXml



- Los estudiantes:
 - Crearán un **DataSet**
 - Cargarán un **DataSet** desde un archivo XML
 - Lo mostrarán en un **DataGrid**
- Tiempo: 5 Minutos

Introducción

En esta práctica, aprenderemos a utilizar el método **ReadXml**.

Los archivos de esta práctica se encuentran en los proyectos Demo10CS y Demo10VB que se pueden encontrar dentro del fichero demos10.zip

🔗 Ejecutar la práctica

1. Abrir el archivo Employees.xml en Visual Studio .NET. En los siguientes pasos, crearemos un formulario Web para mostrar los datos que se encuentran en Employees.xml.
2. Crear un nuevo formulario Web Form en el proyecto Mod12VB o Mod12CS denominado ReadXmlForm.aspx.
3. Arrastrar un control **DataGrid** al formulario Web Form. Establecer su propiedad **ID** a **dgEmployees**.
4. En el procedimiento de evento **Page_Load**, crear un **DataSet**, invocar **ReadXml** para leer los datos XML del archivo Employees.xml en el **DataSet**, y vincular el **DataGrid** al **DataSet**.

El código debería ser similar al siguiente:

Visual Basic .NET

```
Dim ds As New DataSet()  
ds.ReadXml(Server.MapPath("Employees.xml"))  
dgEmployees.DataSource = ds  
dgEmployees.DataBind()
```

C#

```
DataSet ds = new DataSet();  
ds.ReadXml(Server.MapPath("Employees.xml"));  
dgEmployees.DataSource = ds;  
dgEmployees.DataBind();
```

5. Generar y examinar la página.

Crear datos XML anidados

- De forma predeterminada, la salida de DataTable es secuencial
- Para anidar XML, anidar DataRelation

```
Dim dr As New DataRelation _
    ("name", parentCol, childCol)
dr.Nested = True
ds.Relations.Add(dr)
```

Secuencial

```
<Title name="title1" />
<Title name="title2" />
<Title name="title3" />
<Publisher name="pub1" />
<Publisher name="pub2" />
```

```
DataRelation dr = new
    DataRelation("name",
        parentCol,
        childCol);
dr.Nested = true;
ds.Relations.Add(dr);
```

Anidado

```
<Publisher name="pub1" >
    <Title name="title1" />
    <Title name="title3" />
</Publisher>
<Publisher name="pub2" >
    <Title name="title2" />
</Publisher>
```

Introducción

En una representación relacional de datos, las tablas individuales contienen filas relacionadas entre sí utilizando una columna o un conjunto de columnas. En el **DataSet** de ADO.NET, la relación entre tablas se implementa utilizando una **DataRelation**.

DataRelation

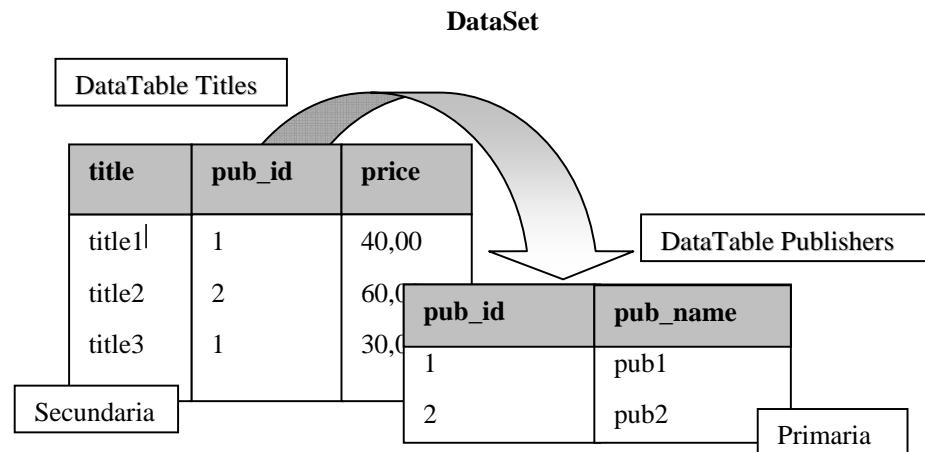
Cuando creamos una **DataRelation**, las relaciones primario-secundario se gestionan únicamente mediante la relación entre las filas y las columnas. Tablas y columnas son entidades distintas. En la representación jerárquica de datos que proporciona XML, las relaciones primario-secundario están representadas por elementos primarios que contienen elementos secundarios anidados.

Nota Cuando se utilizan relaciones anidadas, un elemento secundario sólo puede tener un elemento primario.

Para facilitar la anidación de objetos secundarios cuando un **DataSet** está sincronizado con un **XmlDataDocument**, o cuando está escrito como datos XML utilizando **WriteXml**, el **DataRelation** expone una propiedad **Nested**.

Si se establece a **true** la propiedad **Nested** de una **DataRelation**, las filas secundarias de la relación serán anidadas en la columna primaria cuando el **DataSet** esté escrito como datos XML o se sincronice con un **XmlDataDocument**. El valor predeterminado de la propiedad **Nested** del objeto **DataRelation** es **false**.

En la siguiente ilustración de un **DataSet**, veremos cómo escribir el código cuando la propiedad **Nested** del **DataRelation** está establecida a **false**, además cuando la propiedad **Nested** del **DataRelation** está establecida a **true**. También veremos la salida del resultado de invocar **WriteXml** en el **DataSet**.



El siguiente código muestra cómo establecer la propiedad **Nested** de **DataRelation** a **false**:

Visual Basic .NET

```
Dim ds As New DataSet()
'fill the DataSet
...
Dim parentCol As DataColumn = _
    ds.Tables("Publishers").Columns("pub_id")
Dim childCol As DataColumn = _
    ds.Tables("Titles").Columns("pub_id")
Dim dr As New DataRelation _
    ("TitlePublishers", parentCol, childCol)
ds.Relations.Add(dr)
ds.WriteXml(Server.MapPath("PubTitlesNotNested.xml"), _
    XmlWriteMode.IgnoreSchema)
```

C#

```
DataSet ds = new DataSet();
//fill the DataSet
...
DataColumn parentCol =
    ds.Tables["Publishers"].Columns["pub_id"];
DataColumn childCol= ds.Tables["Titles"].Columns["pub_id"];
DataRelation dr = new DataRelation ("TitlePublishers",
    parentCol, childCol);
ds.Relations.Add(dr);
ds.WriteXml(Server.MapPath("PubTitlesNotNested.xml"),
    XmlWriteMode.IgnoreSchema);
```

La propiedad **Nested** del objeto **DataRelation** no está establecida a **true** para el anterior **DataSet**; por tanto, los objetos secundarios no se anidarán dentro de los elementos primarios cuando este **DataSet** esté representado como datos XML.

El siguiente ejemplo de XML muestra el resultado que resultará de invocar **WriteXml** en el **DataSet**:

```
<?xml version = "1.0" standalone = "yes"?>
<NewDataSet>
  <Titles>
    <title>title1</title>
    <pub_id>1</pub_id>
    <price>40.00</price>
  </Titles>
  <Titles>
    <title>title2</title>
    <pub_id>2</pub_id>
    <price>60.00</price>
  </Titles>
  <Titles>
    <title>title3</title>
    <pub_id>1</pub_id>
    <price>30.00</price>
  </Titles>
  <Publishers>
    <pub_id>1</pub_id>
    <pub_name>pub1</pub_name>
  </Publishers>
  <Publishers>
    <pub_id>2</pub_id>
    <pub_name>pub2</pub_name>
  </Publishers>
</NewDataSet>
```

Los elementos **Titles** y **Publishers** se muestran como elementos secuenciales. Para que los elementos **Titles** aparezcan como secundarios de sus respectivos elementos primarios, la propiedad **Nested** de **DataRelation** debería estar establecida a **true** y deberíamos añadir el siguiente código:

Visual Basic .NET

```
...
Dim dr As New DataRelation _
  ("TitlePublishers", parentCol, childCol)
dr.Nested = True
ds.Relations.Add(dr)
ds.WriteXML(Server.MapPath("PubTitlesNested.xml"), _
  XmlWriteMode.IgnoreSchema)
```

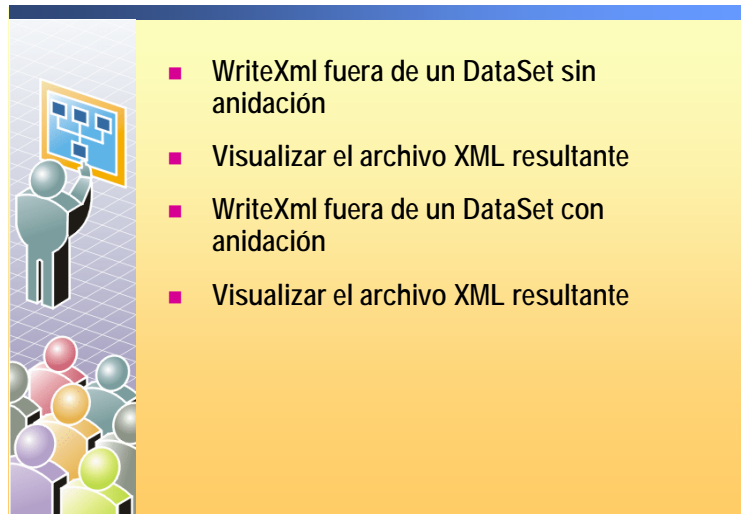
C#

```
...
DataRelation dr = new DataRelation("TitlePublishers",
  parentCol, childCol);
dr.Nested = true;
ds.Relations.Add(dr);
ds.WriteXML(Server.MapPath("PubTitlesNested.xml"),
  XmlWriteMode.IgnoreSchema);
```

El siguiente XML muestra el aspecto que tendría la salida resultante con los elementos **Titles** anidados dentro de sus respectivos elementos primarios:

```
<?xml version = "1.0" standalone = "yes"?>
<NewDataSet>
  <Publishers>
    <pub_id>1</pub_id>
    <pub_name>pub1</pub_name>
    <Titles>
      <title>title1</title>
      <pub_id>1</pub_id>
      <price>40.00</price>
    </Titles>
    <Titles>
      <title>title3</title>
      <pub_id>1</pub_id>
      <price>30.00</price>
    </Titles>
  </Publishers>
  <Publishers>
    <pub_id>2</pub_id>
    <pub_name>pub2</pub_name>
    <Titles>
      <title>title2</title>
      <pub_id>2</pub_id>
      <price>60.00</price>
    </Titles>
  </Publishers>
</NewDataSet>
```

Demostración: crear XML anidado



Introducción

En esta demostración, aprenderemos a crear un documento XML anidado.

Los archivos de esta demostración se encuentran en los proyectos Demo10CS y Demo10VB que se pueden encontrar dentro del fichero demos10.zip

✍ Ejecutar la demostración

1. Abrir la página SaveNestedXML.aspx en Visual Studio .NET.
2. Visualizar la página de código subyacente, explicar el código y anotar lo siguiente:
 - a. En la función **CreateDataSet**, se ha creado un **DataSet** con dos **DataTables**.
 - b. En la función **MakeDataRelation**, se ha creado una **DataRelation** entre las dos tablas, estableciendo la propiedad **Nested** a **True** o **False**, dependiendo del argumento a la función.
 - c. En el procedimiento de evento **cmdSave_Click**, se ha creado la **DataRelation** con **Nested** establecido a **false**, y el **DataSet** se ha escrito en un archivo XML.
 - d. En el procedimiento de evento **cmdSaveNested_Click**, se ha creado la **DataRelation** con la propiedad **Nested** establecida a **true**, y el **DataSet** se ha escrito a un archivo XML.
3. Generar y examinar la página SaveNestedXML.aspx.
4. Hacer clic en **Save as XML**, y clic en el hipervínculo **View XML**.

Éstos son los datos XML creados desde el **DataSet** con la propiedad **Nested** establecida a **false**. Todos los elementos **Titles** están listados, seguidos de los elementos **Publishers**.
5. Hacer clic en **Save as Nested XML**, y clic en el hipervínculo **View Nested XML**.

Éstos son los datos XML que no han creado desde el DataSet con la propiedad Nested establecida a **True**. Los elementos Titles están anidados dentro de los elementos Publishers relacionados.

Lección: trabajar con datos XML

- Descripción de la sincronización de un **DataSet** con un **XmlDataDocument**
- Cómo sincronizar un **DataSet** con un **XmlDataDocument**
- Trabajar con un **XmlDataDocument**
- Transformar datos XML con XSLT
- Demostración: transformar datos con XSLT

Introducción

La clase **XmlDataDocument** permite que los documentos XML sean almacenados, recuperados y manipulados mediante un **DataSet** relacional. **XmlDataDocument** tiene una estrecha afiliación con la clase **DataSet**, que proporciona una vista relacional del documento XML cargado. Los cambios que se realicen al **XmlDataDocument** se reflejan en el **DataSet** y viceversa.

De modo similar, para transformar el contenido de un documento fuente XML en otro formato, como XML o HTML, podemos utilizar una transformación XSLT.

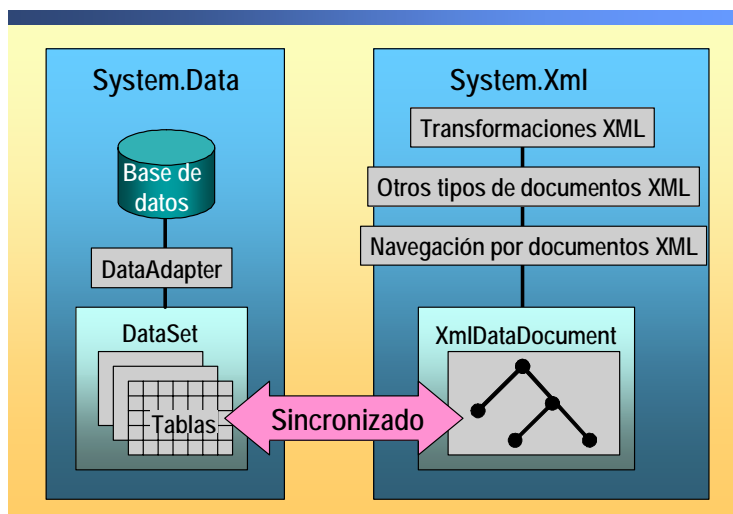
En esta lección, estudiaremos cómo sincronizar un **DataSet** con un **XmlDataDocument**. También aprenderemos a utilizar **XmlDataDocument**. Finalmente, estudiaremos cómo transformar datos XML utilizando el objeto **XsltTransform**.

Objetivos de la lección

En esta lección, aprenderemos a:

- Identificar las diferentes formas disponibles para sincronizar un **DataSet** con un **XmlDataDocument**.
- Identificar cómo sincronizar un **DataSet** con un **XmlDataDocument**.
- Utilizar un **XmlDataDocument**.
- Transformar datos XML utilizando el objeto **XsltTransform**.

Sincronización de un DataSet con un XmlDocument



Introducción

Los **DataSets** proporcionan una representación relacional de los datos. Para el acceso jerárquico a datos, podemos utilizar las clases XML disponibles en el .NET Framework. Antes, las representaciones jerárquicas y relacionales de datos se utilizaban por separado. Sin embargo, el .NET Framework permite el acceso síncrono en tiempo real a las representaciones relacionales y jerárquicas de datos mediante el objeto **DataSet** y el objeto **XmlDataDocument**, respectivamente.

Un único conjunto de datos

Cuando un **DataSet** está sincronizado con un **XmlDataDocument**, ambos objetos trabajan con un único conjunto de datos. Esto significa que si se modifica el **DataSet**, el cambio se reflejará en el **XmlDataDocument**, y viceversa.

La relación entre el **DataSet** y el **XmlDataDocument** crea gran flexibilidad permitiendo que una sola aplicación, utilizando un solo conjunto de datos, acceda a toda la familia de servicios creados alrededor del **DataSet**.

Sincronizar un **XmlDataDocument** con un **DataSet** conserva la fidelidad de un documento XML. Si el **DataSet** se puebla desde un documento XML utilizando **ReadXml**, los datos pueden ser muy distintos del documento XML original cuando los datos son escritos de nuevo como un documento XML utilizando **WriteXml**. Los datos pueden ser diferentes porque el **DataSet** no mantiene el formato, como los espacios en blanco, o la información jerárquica, como el orden de los elementos, del documento XML original. El **DataSet** tampoco contiene los elementos del documento XML que han sido ignorados porque no cumplían el esquema del **DataSet**. Sincronizar un **XmlDataDocument** con un **DataSet** permite mantener el formato y la estructura jerárquica de los elementos del documento XML original en el **XmlDataDocument**, mientras que el **DataSet** contiene únicamente los datos y la información de esquema apropiados para el **DataSet**.

Cómo sincronizar un DataSet con un XmlDocument

■ Almacenar datos XML en un XmlDocument

```
Dim objXmlDataDoc As New XmlDocument()
objXmlDataDoc.Load(Server.MapPath ("file.xml"))
-or-
objXmlDataDoc.DataSet.ReadXml(Server.MapPath ("file.xml"))
```

```
XmlDataDocument objXmlDataDoc = new XmlDocument();
objXmlDataDoc.Load(Server.MapPath ("file.xml"));
-or-
objXmlDataDoc.DataSet.ReadXml(Server.MapPath ("file.xml"));
```

■ Almacenar un DataSet en un XmlDocument

```
Dim ds As New DataSet()
'fill in ds
Dim objXmlDataDoc As New XmlDocument(ds)
```

```
DataSet ds = new DataSet();
//fill in ds
objXmlDataDoc = new XmlDocument(ds);
```

Introducción

Existen dos modos de sincronizar un **DataSet** con un **XmlDataDocument**. Podemos:

- Almacenar datos XML en un **XmlDataDocument**.
- Almacenar un **DataSet** en un **XmlDataDocument**.

Almacenar datos XML en un XmlDocument

El siguiente código de ejemplo muestra cómo almacenar datos XML en un **XmlDataDocument**:

Visual Basic .NET

```
Dim objXmlDataDoc As New XmlDocument()
objXmlDataDoc.Load(Server.MapPath("file.xml"))
```

C#

```
XmlDataDocument objXmlDataDoc = new XmlDocument();
objXmlDataDoc.Load(Server.MapPath("file.xml"));
```

La primera línea del código anterior crea un objeto **XmlDataDocument**, y la segunda línea carga el archivo XML en el objeto **XmlDataDocument**.

También podemos almacenar datos XML en un **XmlDataDocument** utilizando una línea de código como muestran los siguientes ejemplos:

Visual Basic .NET

```
objXmlDataDoc.DataSet.ReadXml(Server.MapPath("file.xml"))
```

C#

```
objXmlDataDoc.DataSet.ReadXml(Server.MapPath("file.xml"));
```

Almacenar un DataSet en un XmlDocument

El siguiente código de ejemplo muestra cómo almacenar un **DataSet** en un **XmlDataDocument**:

Visual Basic .NET

```
Dim ds As New DataSet()  
'fill in ds  
...  
Dim objXmlDataDoc As New XmlDocument(ds)
```

C#

```
DataSet ds = new DataSet();  
//fill in ds  
...  
XmlDataDocument objXmlDataDoc = new XmlDocument(ds);
```

La primera línea del código anterior crea un nuevo **DataSet** denominado **ds**. La última línea crea un objeto denominado **XmlDataDocument** y pasa **ds**, un **DataSet**, como parámetro. El proceso de rellenar **ds** se ha omitido del ejemplo.

Trabajar con un XmlDocument

- Mostrar datos en un control enlazado a lista


```
dg.DataSource = objXmlDataDoc.DataSet
```

```
dg.DataSource = objXmlDataDoc.DataSet;
```
- Extraer filas del DataSet como XML


```
Dim elem As XmlElement
elem = objXmlDataDoc.GetElementFromRow _
(ds.Tables(0).Rows(1))
```

```
XmlElement elem;
elem = objXmlDataDoc.GetElementFromRow(ds.Tables[0].Rows[1]);
```

 - XmlDocument hereda de XmlDocument
- Aplicar una transformación XSLT
 - Objeto XsltTransform

Introducción

El **DataSet** representa una fuente de datos relacional en ADO.NET. El **XmlDocument** implementa el DOM en XML, y el **XmlDataDocument** unifica ADO.NET y XML representando datos relacionales de un **DataSet** y sincronizando esos datos con el modelo de documento XML.

Mostrar datos en un control enlazado a lista

El control **DataGrid** muestra todas las filas de la tabla del **DataSet**. El siguiente código muestra cómo asignar el objeto **DataSet** (**objXmlDataDoc.DataSet**) al control **DataGrid** (**dg**):

Visual Basic .NET

```
dg.DataSource = objXmlDataDoc.DataSet
```

C#

```
dg.DataSource = objXmlDataDoc.DataSet;
```

Extraer filas del DataSet

Para extraer filas individuales como XML, necesitamos consultar el **DataSet**. Para consultar el **DataSet**, utilizamos el método **GetElementFromRow**. El siguiente código muestra cómo el método **GetElementFromRow** de **XmlDataDocument** devuelve un objeto **XmlElement**:

Visual Basic .NET

```
Dim elem As XmlElement
elem = objXmlDataDoc.GetElementFromRow _
(ds.Tables(0).Rows(1))
```

C#

```
XmlElement elem;
elem = objXmlDataDoc.GetElementFromRow(ds.Tables[0].Rows[1]);
```

Utilizar métodos XML DOM

El .NET Framework implementa el XML DOM para proporcionar acceso a datos de documentos XML y proporcionar acceso a las clases adicionales para poder leer, escribir y navegar en documentos XML. **XmlDataDocument** proporciona un acceso relacional a datos con su capacidad de sincronizar con los datos relacionales del **DataSet**.

La clase **XmlDataDocument** extiende la clase **XmlDocument**. La clase **XmlDocument** implementa el DOM; por tanto, nos permite cargar datos relacionales o datos XML. **XmlDataDocument** también nos permite manipular esos datos utilizando el DOM.

Aplicar una transformación XSLT

Si los datos se almacenan en una estructura relacional y deseamos que sean la entrada en una transformación XSLT, podemos cargar los datos relacionales en un **DataSet** y asociarlos con el **XmlDataDocument**.

Tomando datos relacionales, cargándolos en un **DataSet** y utilizando la sincronización dentro del **XmlDataDocument**, los datos relacionales pueden sufrir transformaciones XSLT. El objeto **XsltTransform** transforma datos XML utilizando una hoja de estilo XSLT.

Transformar datos XML con XSLT

■ Crear un XmlDocument

```
Dim ds As New DataSet()
'fill in DataSet
...
Dim xmlDoc As New XmlDocument(ds)
```

■ Crear el objeto XSLTransform e invocar el método Transform

```
Dim xslTran As New XslTransform()
xslTran.Load(Server.MapPath("PubTitles.xsl"))
Dim writer As New XmlTextWriter _
    (Server.MapPath("PubTitles_output.html"), _
    System.Text.Encoding.UTF8)
xslTran.Transform(xmlDoc, Nothing, writer)
writer.Close()
```

[Código de ejemplo de #](#)

Introducción

El objetivo de la transformación XSLT es transformar el contenido de un documento XML origen en otro documento diferente en formato o estructura. Por ejemplo, transformar XML en HTML para utilizarlo en un sitio Web o transformar XML en un documento que contenga únicamente los campos que requiere una aplicación.

En el .NET Framework, la clase **XslTransform** es el procesador XSLT que transforma un documento XML en otro.

Crear XmlDocument

Antes de transformar datos XML, es necesario crear un DataSet y un objeto **XmlDocument**:

Visual Basic .NET

```
Dim ds As New DataSet()
'fill in DataSet
...
Dim xmlDoc As New XmlDocument(ds)
```

C#

```
DataSet ds = new DataSet();
//fill in DataSet
...
XmlDocument xmlDoc = new XmlDocument(ds);
```

La primera línea del código anterior crea el DataSet. La siguiente línea de código (el código no se muestra; en lugar de ello, el comentario está presente) rellena el DataSet. La última línea del código anterior crea un objeto **XmlDocument** denominado **xmlDoc** y pasa un parámetro a **xmlDoc**, el DataSet **ds**.

Crear el objeto **XslTransform** e invocar el método **Transform**

Los siguientes pasos muestran el proceso de transformar datos XML creando un objeto **XslTransform** e invocando el método **Transform**:

1. Crear un objeto **XslTransform**:

Visual Basic .NET

```
Dim xslTran As New XslTransform()
```

C#

```
XslTransform xslTran = new XslTransform();
```

2. Utilizar el método **Load** para cargar el archivo de hoja de estilo .xsl para la transformación:

Visual Basic .NET

```
xslTran.Load(Server.MapPath("PubTitles.xsl"))
```

C#

```
xslTran.Load(Server.MapPath("PubTitles.xsl"));
```

3. Crear un objeto **XmlTextWriter** para la salida del documento:

Visual Basic .NET

```
Dim writer As New XmlTextWriter _  
    (Server.MapPath("PubTitles_output.html"), _  
    System.Text.Encoding.UTF8)
```

C#

```
XmlTextWriter writer = new XmlTextWriter  
    (Server.MapPath("PubTitles_output.html"),  
    System.Text.Encoding.UTF8);
```

4. Utilizar el método **Transform** del objeto **XslTransform** para transformar los datos. El método **Transform** tiene múltiples sobrecargas y puede gestionar distintos tipos de entradas y salidas.

La variable **XmlDoc** del tipo **XmlDataDocument** es uno de los parámetros que se pasan al método **Transform**:

Visual Basic .NET

```
xslTran.Transform(xmlDoc, Nothing, writer)
```

C#

```
xslTran.Transform(xmlDoc, null, writer);
```

5. Cerrar **XmlTextWriter**:

Visual Basic .NET

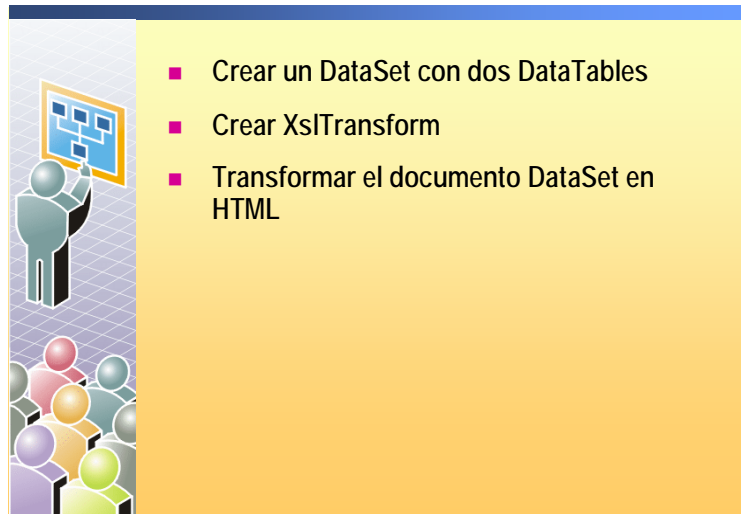
```
writer.Close()
```

C#

```
writer.Close();
```

Nota El proceso de transformación está especificado por la Recomendación XSLT versión 1.0 del W3C. Para más información, ver www.w3c.org/TR/xslt.

Demostración: transformar datos con XSLT



Introducción

En esta demostración, veremos cómo transformar datos utilizando el objeto **XslTransform**.

Los archivos de esta demostración se encuentran en los proyectos Demo10CS y Demo10VB que se pueden encontrar dentro del fichero demos10.zip

🔗 Ejecutar la demostración

1. Abrir la página TransformPubTitles.aspx.
Los dos controles **DataGrid** muestran las dos **DataTable**s en el **DataSet**.
2. Visualizar la página de código subyacente.
3. Mostrar el código del procedimiento de evento **cmdTransform_Click** y explicar el código:
 - a. El **DataSet** se crea utilizando **CreateDataSet** para crear el **DataSet**, y seguidamente se invoca a **MakeDataRelation** para crear la **DataRelation**. No obstante, para que el mapping **XmlDataDocument** funcione, debe agregarse al **DataSet** la tabla primaria, **Publishers**, antes de la tabla **Titles**.
 - b. Se crea un **XmlDataDocument** desde el **DataSet**.
 - c. Se crea un objeto **XslTransform** y se carga con la hoja de estilo PubTitles.xsl.
 - d. Se invoca el método **Transform** del objeto **XslTransform** para aplicar la hoja de estilo al **XmlDocument**.

Nota El método **Transform** únicamente puede enviar la salida a un **XmlReader**, un **TextReader** o a objetos **XmlWriter**.

4. Visualizar la página en un navegador.
5. Hacer clic en **Transform Data**, y clic en el hipervínculo **View Transform Output**.

Ésta es la página HTML creada a partir de los datos del **DataSet**.

6. Abrir la página **Transform.aspx** en Visual Studio .NET.
7. Visualizar la página de código subyacente.
8. Mostrar el código en el procedimiento de evento **cmdTransform_Click** y explicar el código:
 - a. Se crea el **DataSet** invocando **CreateCustOrdersDataSet** para crear el **DataSet** y el **DataRelation**.
 - b. Se crea un **XmlDataDocument** a partir del **DataSet**.
 - c. Se crea un objeto **XslTransform** y se carga con la hoja de estilo **CustomerOrders.xslt**.
 - d. Se invoca el método **Transform** del objeto **XslTransform** para aplicar la hoja de estilos al **XmlDocument**.
9. Abrir la hoja de estilos **PubTitles.xsl** para mostrar cómo funciona.

Lección: uso del control de Servidor Web XML

- ¿Qué es el control de Servidor Web XML?
- Cargar y guardar datos XML
- Demostración: uso del control de Servidor Web XML

Introducción

La información en un archivo XML es plana, únicamente contiene los datos y no indica cómo formatearlos o mostrarlos. Para mostrar datos XML en la página de un formulario Web Form, debemos proporcionar la información de formato y presentación.

En esta lección, aprenderemos cómo mostrar, cargar y guardar datos XML.

Objetivos de la lección

En esta lección, aprenderemos a:

- Describir el control de Servidor Web XML.
- Cargar y guardar datos XML.

¿Qué es el control de Servidor Web XML?

- Escribe a un documento XML
- Escribe el resultado de una transformación XSLT a una página Web

```
<asp:Xml id="Xml1"
  Document="XmlDocument object to display"
  DocumentContent="String of XML"
  DocumentSource="Path to XML Document"
  Transform="XslTransform object"
  TransformSource="Path to XSL Document"
  runat="server"/>
```

Introducción

Para presentar datos XML en una página de un formulario Web Form, debemos especificar las etiquetas, como <TABLE>, <P>, o cualquier otra etiqueta que deseemos utilizar para mostrar los datos. También debemos proporcionar instrucciones sobre cómo los datos del archivo XML se muestran en estas etiquetas; por ejemplo, si cada elemento del archivo XML debería mostrarse como una fila o una columna de tabla, etc.

Lenguaje de transformación XSLT

Una forma de proporcionar todas estas instrucciones es utilizar el lenguaje de transformación XSLT y crear archivos XSLT. Una vez dispongamos de las transformaciones XSLT, debemos aplicarlas al archivo XML. El resultado es un nuevo archivo con la información XML formateada según el archivo de transformación.

Utilizar el control de Servidor Web XML para escribir un documento XML

Podemos utilizar el control de Servidor Web XML para escribir un documento XML, o para escribir el resultado de una transformación XSLT, en una página Web. El resultado XML aparece en la ubicación del control de la página Web.

La información del XML y de XSLT puede encontrarse en documentos externos, o podemos incluir el XML en línea. Existen dos formas de referenciar documentos externos utilizando las configuraciones de propiedades del control de Servidor Web XML. Podemos proporcionar una ruta al documento XML de la etiqueta del control, o podemos cargar los documentos XML y XSLT como objetos y pasarlos al control. Si preferimos incluir el XML en línea, debemos escribirlo entre las etiquetas de apertura y cierre del control.

El siguiente código de ejemplo muestra cómo utilizar el control de Servidor Web XML para mostrar el contenido de un documento XML o el resultado de una transformación XSLT:

```
<asp:Xml id="Xml1"
  Document="XmlDocument object to display"
  DocumentContent="String of XML"
  DocumentSource="Path to XML Document"
  Transform="XslTransform object"
  TransformSource="Path to XSL Transform Document"
  runat="server">
```

Cargar y guardar datos XML

- Control de Servidor Web XML (en el formulario Web Form)

```
<asp:Xml id="xmlCtl" runat="server" />
```

- Cargar datos dinámicamente (en la página de código subyacente)

```
xmlCtl.Document.Load(Server.MapPath("text.xml"))
```

```
xmlCtl.Document.Load(Server.MapPath("text.xml"));
```

- Guardar datos (en la página de código subyacente)

```
xmlCtl.Document.Save(Server.MapPath("text.xml"))
```

```
xmlCtl.Document.Save(Server.MapPath("text.xml"));
```

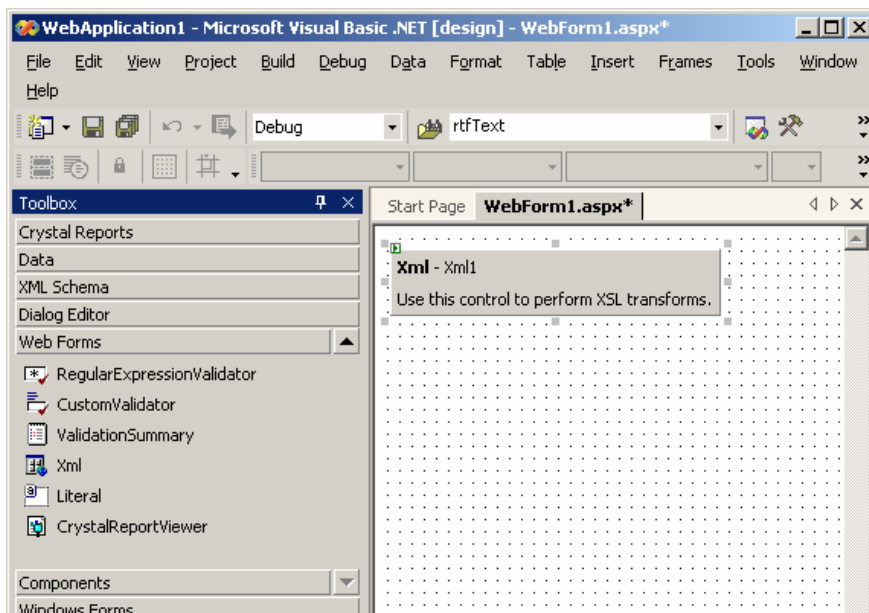
Introducción

Antes de cargar y guardar datos XML en una aplicación Web, debemos agregar el control de servidor Web XML a la página del formulario Web Form, en la ubicación donde deseamos que aparezca el resultado.

Agregar un control de Servidor Web XML a la página de un formulario Web Form

Existen dos formas de agregar un control de servidor Web XML a la página de un formulario Web Form:

1. Arrastrar un control **XML** desde la ficha **Web Forms** del Cuadro de herramientas a la vista de Diseño, como muestra la siguiente ilustración.



2. Para agregar un control de servidor Web XML programáticamente, en la vista HTML, agregar la siguiente línea de código:

```
<asp:Xml id="xmlCtl" runat="server" />
```

Cargar datos XML en el control de Servidor Web XML

Existen varias formas de cargar datos XML en una aplicación Web. Podemos:

- Proporcionar una ruta a un documento XML externo utilizando la propiedad **DocumentSource**.
- Cargar un documento XML como un objeto y pasarlo al control, utilizando el método **Load** del evento **Page_Load**, y asignar el documento a la propiedad **Document** del control XML.
- Incluir el contenido XML en línea, entre las etiquetas de apertura y cierre del control de Servidor Web XML.

Proporcionar una ruta a un documento XML externo

Para proporcionar una ruta a un documento XML externo, seguir estos pasos:

1. Establecer la propiedad **DocumentSource** del control de Servidor Web XML a la ruta del documento fuente XML.
2. El documento XML se escribirá directamente al flujo de salida a menos que también especifiquemos la propiedad **TransformSource**. **TransformSource** debe ser un documento válido de transformación XSLT, que se utilizará para transformar el documento XML antes de que su contenido se escriba en el flujo de salida. El siguiente ejemplo muestra cómo hacer referencia a documentos origen utilizando una ruta relativa:

```
<body>
  <h3>XML Example</h3>
  <form runat="server">
    <asp:Xml id="xml1" DocumentSource="MySource.xml"
      TransformSource="MyStyle.xsl" runat="server" />
  </form>
</body>
```

Cargar un documento XML como un objeto y pasarlo al control

Para cargar un documento XML como un objeto y pasarlo al control, seguir estos pasos:

1. En el menú **Ver**, hacer clic en **Código**. En el Editor de código, buscar el procedimiento de evento **Page_Load**.
2. Agregar código para cargar el documento fuente XML, y asignar la fuente a la propiedad **Document** del control. Por ejemplo:

Visual Basic .NET

```
Private Sub Page_Load(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load
  Dim xmlDoc As System.Xml.XmlDocument = _
    New System.Xml.XmlDocument()
  xmlDoc.Load(Server.MapPath("MySource.xml"))
  Dim xslTran As System.Xml.Xsl.XslTransform = _
    New System.Xml.Xsl.XslTransform()
  xslTran.Load(Server.MapPath("MyStyle.xsl"))
  Xml1.Document = xmlDoc
  Xml1.Transform = xslTran
End Sub
```



```

C#      private void Page_Load(object sender, System.EventArgs e)
        {
            System.Xml.XmlDocument xmlDoc = new
                System.Xml.XmlDocument();
            xmlDoc.Load(Server.MapPath("MySource.xml"));
            System.Xml.Xsl.XslTransform xslTran = new
                System.Xml.Xsl.XslTransform();
            xslTran.Load(Server.MapPath("MyStyle.xsl"));
            Xml1.Document = xmlDoc;
            Xml1.Transform = xslTran;
        }

```

Incluir el contenido XML en línea

Para incluir el contenido XML en línea, seguir estos pasos:

1. En la vista HTML, buscar las etiquetas `<asp:xml>` y `</asp:xml>`.
2. Agregar nuestro código XML entre ambas etiquetas. Por ejemplo:

```

<asp:xml TransformSource="MyStyle.xsl" runat="server">
    <clients>
        <name>Frank Miller</name>
        <name>Judy Lew</name>
    </clients>
</asp:xml>

```

Guardar datos XML

Podemos guardar los datos XML utilizando el método **Save**, como muestra el siguiente código de ejemplo:


Visual Basic .NET

```
XmlCtl.Document.Save(Server.MapPath("xmlResult.xml"))
```

C#

```
XmlCtl.Document.Save(Server.MapPath("xmlResult.xml"));
```

Demostración: uso del control de servidor Web XML



- Agregar el control de Servidor Web XML a un formulario Web Form
- Establecer la propiedad **DocumentSource** para leer un archivo XML
- Ver el resultado
- Establecer la propiedad **TransformSource** para leer un archivo XSLT
- Ver el resultado

Introducción

En esta demostración, estudiaremos cómo utilizar el control de Servidor Web XML.

Los archivos de esta demostración se encuentran en los proyectos Demo10CS y Demo10VB que se pueden encontrar dentro del fichero demos10.zip

El código completo para esta demostración se encuentra en el archivo UseXmlControl.aspx.

⚡ Ejecutar la demostración

1. Abrir el archivo PubTitlesData.xml.
Este archivo contiene los datos que se mostrarán de la página ASPX.
2. Crear un nuevo formulario Web en el proyecto Mod12VB o Mod12CS denominado MyUseXmlControl.aspx.
3. Arrastrar el control **XML** desde el Cuadro de herramientas al formulario Web Form.
4. Establecer la propiedad **DocumentSource** del control **XML** al archivo PubTitlesData.xml.
5. Generar y examinar.
Ésta es la vista predeterminada de los datos según lo establecido por el control. La vista predeterminada de los datos no tiene formato.
6. Establecer la propiedad **TransformSource** del control **XML** al archivo PubTitles.xsl.
7. Generar y examinar la página de nuevo.
Ésta es la vista de los datos según lo establecido por la hoja de estilos **PubTitles.xsl**.