XML.NET

XML antes de .NET.

La tecnología .NET introduce una serie de APIs XML basadas en estándares como DOM, XPath, XSD (XML Schemas), XSLT y SOAP. Tales APIs están compuestas por una serie de clases que pertenecen a varios `namespace` (siendo el contenedor de todos `System.Xml`) y que hacen más fácil, flexible e intuitiva la programación de aplicaciones con soporte XML.

Estas clases son una evolución del modelo MSXML 3.0, que es el modelo XML que se utilizaba en las aplicaciones Microsoft antes de .NET. Los componentes del modelo XML 3.0 se encuentran en la librería `MSXML3.dll` y siguen el estándar DOM.
Aunque lo lógico es utilizar `MSXML3.dll` para aplicaciones no .NET y las clases del `namespace System.Xml` para aplicaciones .NET, es posible utilizar `MSXML3.dll` desde .NET a través de la interoperabilidad COM. Es decir, utilizando `Tlbimp` se puede crear una clase `wrapper` para la librería `MSXML3.dll` y luego incluirla en una aplicación C#.

Por ejemplo:

```
...
...
using MSXML3;
DOMDocument30 docu = new DOMDocument30();
if (docu.load("fich1.xml"))
{
//tratamiento del documento
}
...
...
```

Al desarrollar aplicaciones .NET, lo más recomendable es trabajar con las clases del `namespace System.Xml` pero existen casos en los que está opción puede no ser la más interesante, como por ejemplo cuando ya se tiene mucho código que no es .NET ya desarrollado y se quiere reutilizar.

El namespace System.Xml.

Las clases XML del Framework .NET están repartidas en varios `namespace` que pertenecen al `namespace System.Xml`.

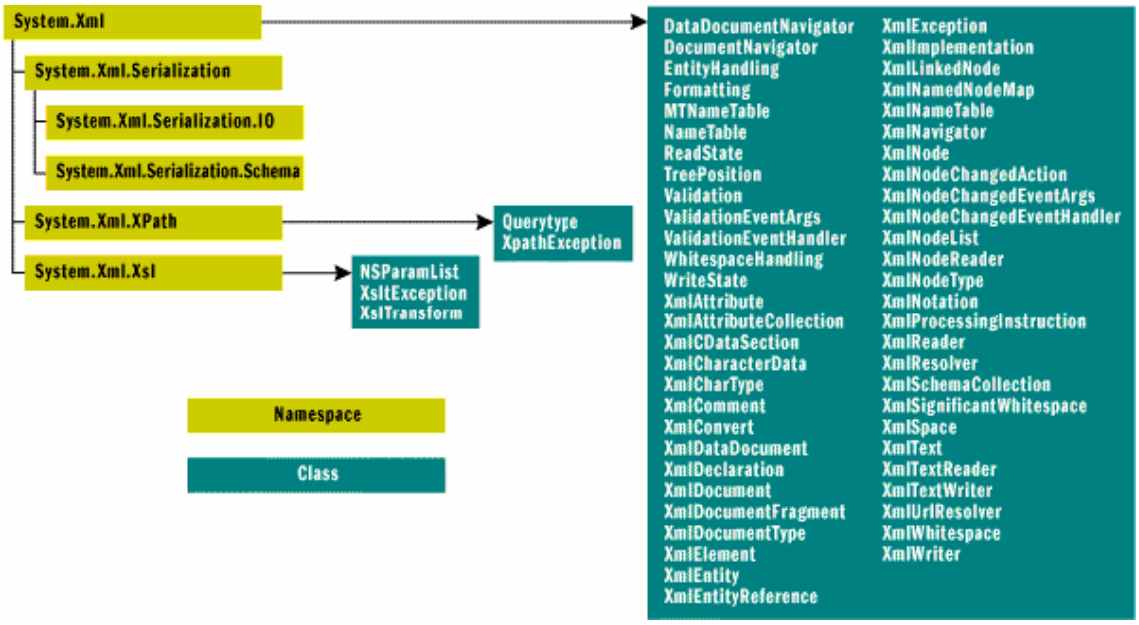


Figura 25.1. Jerarquía XML en .NET.

Todos estos `namespace` se encuentran en la librería `System.Xml.dll`.

XmlReader.

Es una clase base abstracta que ofrece un cursor de tipo `forward-only, read-only` (lectura sólo hacia delante) para los documentos XML. El hecho de que el cursor sea de tipo `forward-only` hace que sea muy rápido.

`XmlReader` permite leer un documento como un stream, sin haberlo cargado completamente en memoria, del mismo modo que SAX. No como el modelo DOM, que para leer un documento y procesarlo exige cargarlo previamente en memoria. No obstante, existe una diferencia importante entre el funcionamiento de `XmlReader` y el modelo SAX:

- Según el modelo SAX, cuando una aplicación utiliza un `XmlReader SAX` para leer un documento XML, la aplicación debe implementar una serie de interfaces (`LexicalHandler`, `ContentHandler`, `ErrorHandler`) a los que va invocando el `XmlReader` según va leyendo el documento XML. A este modelo se le llama `Modelo Push`, porque es el `XmlReader` el que avisa a la aplicación.

- Según el modelo `XmlReader` .NET, cuando una aplicación utiliza un `XmlReader` .NET para leer un documento XML, el objeto `XmlReader` debe implementar una serie de interfaces (`XmlTextReader`, `XmlReader`, `XmlNodeReader`) a las que invocará la aplicación cuando desee, siendo la aplicación la que dispone del control sobre el avance del parseo o análisis del documento XML. A este modelo se le llama `Modelo Pull`, porque es la aplicación la que controla al `XmlReader`.

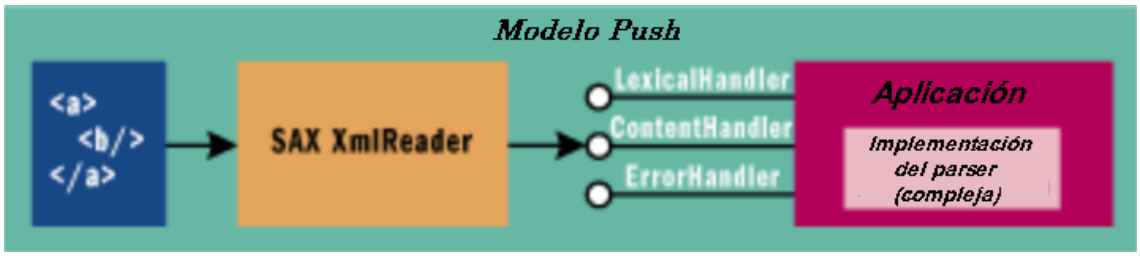


Figura 25.2. `Modelo Push`. corresponde a la lectura y parseo de documentos XML según el modelo SAX.

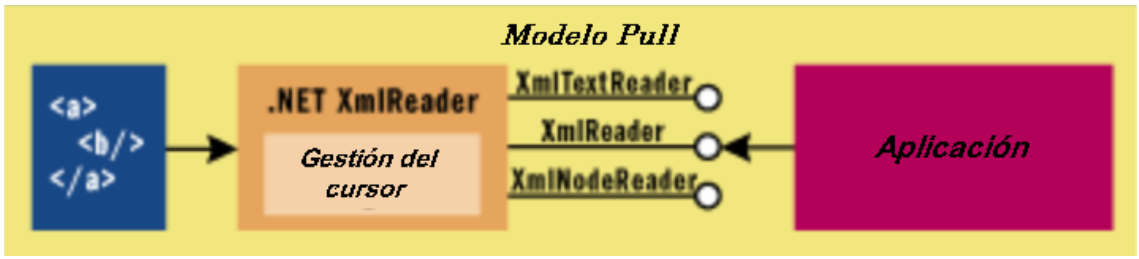


Figura 25.3. `Modelo Pull`. Corresponde a la lectura y parseo de documentos XML mediante un objeto `XmlReader` .NET (que implemente la clase abstracta `XmlReader`).

Varias clases implementan la clase `XmlReader`. Las más significativas son `XmlTextReader` y `XmlNodeReader`. `XmlTextReader` permite leer un stream de tipo texto (un stream XML es eso, al fin y al cabo) y `XmlNodeReader` permite trabajar con un árbol DOM cargado en memoria (correspondiente, por supuesto, a un documento XML, representado por un objeto de la clase `XmlNode`).

XmlWriter.

Es una clase base abstracta que define la funcionalidad básica para generar documentos XML como streams. La generación de un documento utilizando `XmlWriter` .NET es muy similar a hacerlo mediante la clase `XmlWriter` SAX.

`XmlWriter` ofrece métodos para emitir todas las construcciones del `Infoset` estándar en XML (`WriteStartDocument`, `WriteStartElement`, `WriteEndDocument`, `WriteProcessingInstruction`...).

Del mismo modo que ocurre con `XmlReader`, varias clases implementan la clase `XmlWriter`. Éstas son `XmlTextWriter` y `XmlNodeWriter`, similares a sus homónimas `XmlTextReader` y `XmlNodeReader` pero en sentido opuesto (escritura).

XmlNavigator.

Es una clase base abstracta que define la funcionalidad común a cualquier navegador de documentos. Esta clase ofrece un mecanismo genérico de navegación basado en la recomendación XPath 1.0.

La clase `XmlNavigator` soporta rutinas de navegación genéricas, selección de nodos, iteración sobre la selección y otras opciones avanzadas sobre la selección, como copiar, borrar, mover, etc...

Una ventaja frente a DOM es que una implementación de `XmlNavigator` no necesita cargar un documento completo en memoria para manejar una parte del mismo.

Entre los métodos más relevantes ofrecidos por `XmlNavigator` están `Select` y `SelectSingle`, que equivalen a los métodos `selectNodes` y `selectSingleNode` de MSXML 3.0. Ambos métodos aceptan una expresión XPath bien en forma de `string` o como una expresión precompilada, que será evaluada para identificar un conjunto de nodos.

Lectura de documentos XML.

Existen 3 clases derivadas de `XmlReader` que permiten leer documentos XML.

- `XmlTextReader`, que sigue el modelo pull, que es una variante del modelo push de SAX.

- `XmlNodeReader,` que permite leer un documento cargado en memoria, siguiendo el modelo DOM.

- `XmlValidatingReader` que permite validar el documento a leer contra su DTD o su Schema.

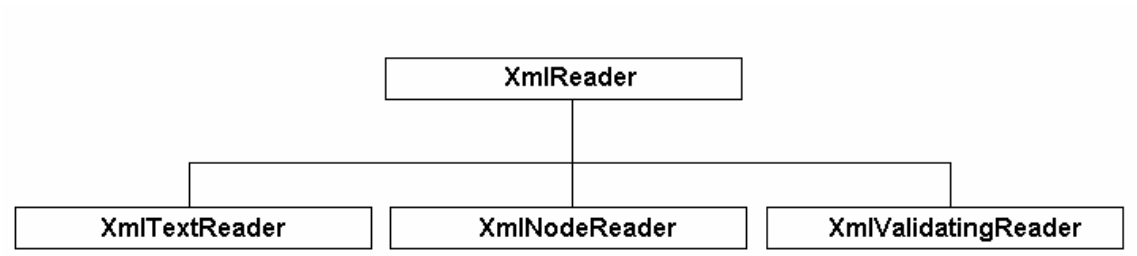


Figura 25.4. `XmlReader` y clases que la implementan.

`XmlTextReader`, `XmlNodeReader` y `XmlValidatingReader`, implementan los métodos declarados por `XmlReader`. `XmlReader` declara métodos y propiedades para:

- Leer Nodos (métodos `Read`, `ReadStartElement`...).

- Obtener información sobre un Nodo (propiedades `Name`, `LocalName`, `NameSpaceURI`, `Value`, `HasAttributes`, `AttributeCount`).

- Expandir referencias a Entidad (método `ResolveEntity`).

- Moverse y obtener información de Atributos (métodos `GetAttribute`, `MoveToAttribute`, `MoveToFirstAttribute`, `MoveToNextAttribute`)

XMLTextReader.

Como ya se ha comentado, `XmlTextReader` implementa los métodos de la clase `XmlReader` y por supuesto, añade nuevos métodos y propiedades. `XmlTextReader` permite analizar o parsear un stream que contenga un documento de tipo texto XML, dividiéndolo en tokens o unidades léxicas que contendrán los distintos elementos del documento.

El tipo de stream se le pasará como parámetro al constructor. `XmlTextReader` tiene tantos constructores como tipos diferentes de streams soporta (streams simples, `strings` de tipo URI con el nombre y la ruta al fichero, objetos `TextReader`).

Entre las características nuevas que añade `XmlTextReader` cabe destacar:

- Soporte para comprobar que el documento XML está bien formado (en base al DTD). `XmlTextReader` no permite validar el documento, sólo comprobar si está bien formado. Si se desea validar el documento se ha de utilizar `XmlValidatingReader`.

- Soporte para resolución personalizada de entidades externas: La resolución personalizada de entidades externas implica crear una clase derivada de `XmlResolver` que implemente los métodos al efecto (y asociarla al `XmlTextReader` mediante la propiedad `XmlResolver`). Trabajar de este modo permite mejorar el rendimiento.

Lectura de un documento XML utilizando XmlTextReader.

Una vez se dispone de un objeto `XmlTextReader` creado (y por tanto asociado a un documento XML), para leer el contenido del documento sólo hay que invocar al método `Read`. El método `Read` lee un nodo cada vez que es invocado, pero no devuelve el nodo, sino que deja al objeto `XmlTextReader` "apuntando" al nodo actual tras la lectura. De este modo, si se desea acceder a la información del nodo, habrá de ser mediante métodos y propiedades del objeto `XmlTextReader` (`NodeType`, `Name`...).

Cuando se inicializa un objeto `XmlTextReader` no hay nodo actual, la primera llamada a `Read` sitúa el objeto `XmlTextReader` en el primer nodo del documento y cada nueva llamada lo va situando en el siguiente nodo.

El orden de lectura de los nodos de un documento XML es de izquierda a derecha y de arriba hacia abajo.

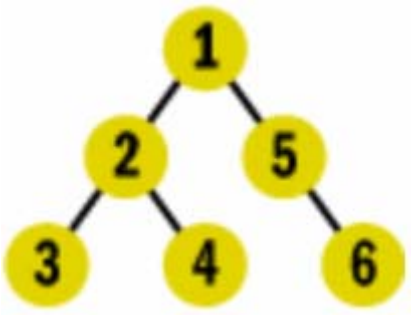


Figura 25.5. Orden de lectura de un documento XML.

El método `Read` es heredado de `XmlReader` e implementado en `XmlTextReader`. No es el único, y existen varios métodos más de lectura heredados de `XmlReader`:

- `ReadStartElement`: este método permite comprobar en la lectura que el nodo actual es un elemento con un nombre específico y una URI específica. En caso de que el nodo actual no tenga el nombre y URI pasados como parámetros a `ReadStartElement` se lanzará una excepción de tipo `XmlException`.

- `ReadString`: Este método, al leer un nodo de tipo texto, devuelve su contenido como un `string`. Por supuesto, al invocarlo se debe saber de qué tipo es el valor del nodo actual (el nodo a leer). Si se desea obtener el contenido de un nodo cuyo tipo sea distinto de `string` no existen para ello métodos del tipo `ReadInt`, `ReadDouble`, etc.... En su lugar ha de utilizarse la clase `XmlConvert`.

  Por ejemplo: `Double precio = XmlConvert.ToDouble(miReader.Read());`

- `GetAttribute`: Los métodos de tipo `Read` leen los nodos de un documento XML, pero no los atributos. Si el nodo actual es un elemento, pueden obtenerse sus atributos invocando al método `GetAttribute`, pasándole como parámetro el nombre o el índice del atributo.

En el siguiente ejemplo, se desarrolla una aplicación de tipo Win Forms llamada `LecturaDocsXML1` que consta de una caja de texto `textBox1`, con la propiedad `MultiLine` a `true`, un botón de pulsación `button1`, con la propiedad `Name = "Leer"` y una lista `listBox1` y que, además,

1) Carga el contenido del fichero `libros.xml` en la caja de texto (en este punto se utilizan clases de manejo de ficheros, no clases XML).
2) Al pulsar el botón `Leer` se crea un objeto `XmlTextReader` con el que se lee un stream asociado al fichero `libros.xml`, añadiendo el valor -propiedad `Value`- de cada Nodo del fichero a la lista.

1) Carga del contenido del fichero `libros.xml` en la caja de texto: Se utilizan las clases `FileStream` para abrir el fichero `libros.xml` y `StreamReader` para leer el contenido del fichero. Estas clases pertenecen al `namespace System.IO` y no tienen nada que ver con XML. Se utilizan en este ejemplo para cargar el fichero `libros.xml` en la caja de texto, haciendo más agradable el ejemplo y mostrando que, al fin y al cabo, un fichero XML no deja de ser un fichero de texto. El código para realizar lo comentado puede incluirse en el método `Form1_Load`, que es el método que se ejecuta al cargar el formulario de la aplicación.

```
...
...
using System.IO;
...
...
private void Form1_Load(object sender, System.EventArgs e)
```

```
{
        FileStream fich;

        fich = new FileStream("..\\..\\libros.xml", FileMode.Open,
        FileAccess.Read);

        StreamReader str = new StreamReader(fich);

        // Situar el puntero de lectura al principio.
        str.BaseStream.Seek(0, SeekOrigin.Begin);

        // Leer el fichero y escribirlo en la caja de texto.
        textBox1.Text = str.ReadToEnd();
        str.Close();
}
...
...
```

El resultado de ejecutar la aplicación, en este momento, será:

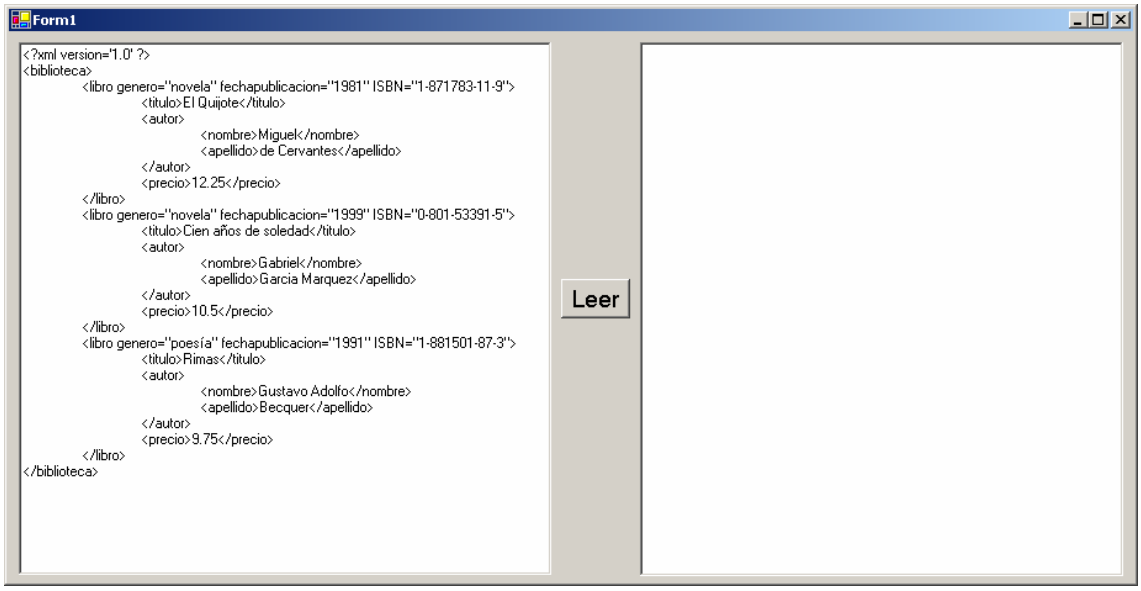


Figura 25.6. Aplicación `LecturaDocsXML1`. Carga, mediante lectura tradicional, del fichero `libros.xml` en la caja de texto.

2) Lectura, utilizando `XmlTextReader`, y adición de los valores de los Nodos a la lista: para este punto se utiliza un objeto de la clase `XmlTextReader`. Al construirlo se le pasa un objeto `FileStream` correspondiente al fichero `libros.xml`. Como puede observarse, la forma de leer un fichero con `XmlTextReader` es similar a la forma de hacerlo con `StreamReader`. La diferencia estriba en la implementación de la lectura, `XmlTextReader` está especializada para leer documentos XML (a `StreamReader` le da igual). El código correspondiente a este punto ha de escribirse en el método de respuesta al evento `Click` sobre el botón `Leer` (`button1`).

```
...
...
using System.IO;
```

```
using System.Xml;
...
...
private void button1_Click(object sender, System.EventArgs e)
{
      FileStream fich;
      fich = new FileStream("..\\..\\libros.xml", FileMode.Open,
      FileAccess.Read);

      XmlTextReader lector1 = new XmlTextReader(fich);

      while (lector1.Read())
      {
            listBox1.Items.Add(lector1.Value);
      }
}
...
...
```

El resultado de ejecutar la aplicación, después de pulsar el botón `Leer` es:

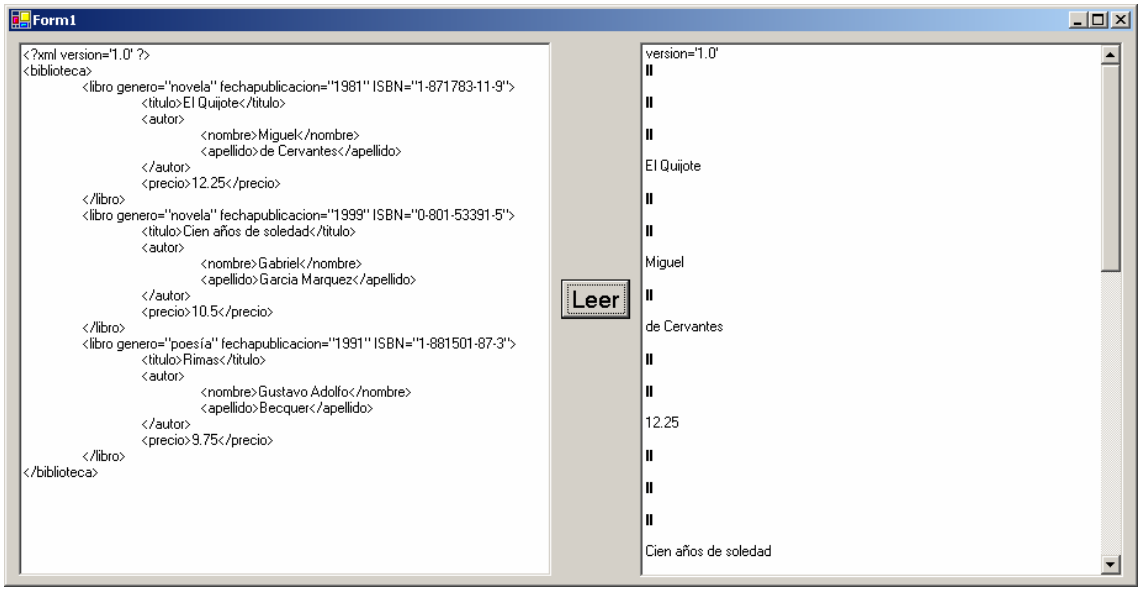


Figura 25.7. Aplicación `LecturaDocsXML1`. Lectura, mediante `XmlTextReader`, del documento `libros.xml`.

Como puede observarse, el contenido de la lista no coincide con el contenido de la caja de texto. El motivo es que, aunque el método `Read` de `XmlTextReader` va leyendo Nodo a Nodo, la propiedad `Value` sólo devuelve lo que se considera es el contenido del nodo.

Por ejemplo: el contenido del nodo `<titulo>El Quijote</titulo>` es "`El Quijote`". En cambio, el contenido del nodo `<libro genero="novela" fechapublicación...>...</libro>` se considera vacío (`Value` devuelve `String.Empty`), porque contiene otros nodos, no un texto.

A continuación se muestra una tabla con los tipos de nodo para los que la propiedad `Value` devuelve un valor. Para el resto de Nodos, devuelve `String.Empty`.

Tipo de Nodo	Valor devuelto por la propiedad Value
`Attribute`	valor del atributo
`CDATASection`	contenido de la sección `CDATA`
`Comment`	contenido del comentario
`DocumentType`	subconjunto (`subset`) interno
`ProcessingInstruction`	contenido completo, excluyendo `target`
`SignificantWhitespace`	El espacio entre las marcas que lo delimitan (modo mixto)
`Text`	contenido del nodo de texto
`Whitespace`	El espacio entre las marcas que lo delimitan
`XmlDeclaration`	contenido de la declaración

Una posible solución al problema anterior puede ser mostrar sólo aquellos valores que tengan sentido. En el documento `libros.xml` los nodos para los que `Value` devolverá un valor con sentido son los de tipo `Text`. Para saber si el Nodo actual es de tipo `Text` puede consultarse la propiedad `NodeType`:

```
private void button1_Click(object sender, System.EventArgs e)
{
      FileStream fich;

            fich = new FileStream("..\\..\\libros.xml", FileMode.Open,
            FileAccess.Read);

            XmlTextReader lector1 = new XmlTextReader(fich);

            while (lector1.Read())
      {
            if (lector1.NodeType == XmlNodeType.Text)
                  listBox1.Items.Add(lector1.Value);
      }
}
```

El resultado ahora es:

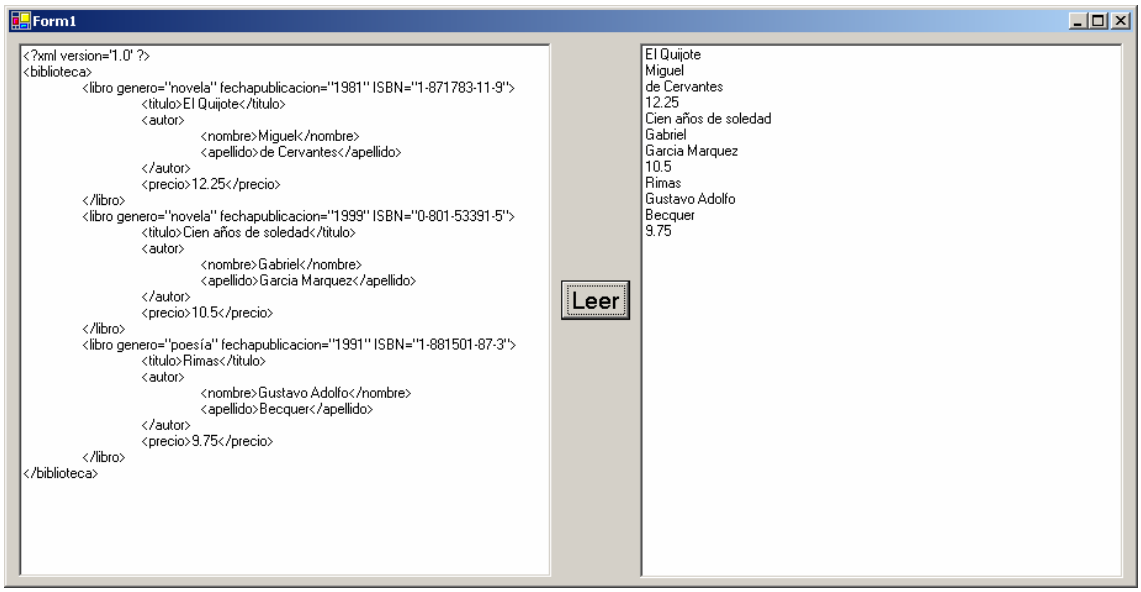


Figura 25.8. Aplicación `LecturaDocsXML1`. Utilización de la propiedad `NodeType` de `XmlTextReader` para mostrar sólo los Nodos de tipo Texto.

Esto ya está algo mejor pero no lo suficiente. ¿Qué sucede con los atributos `genero`, `fechapublicación` e `ISBN`?. El método `Read` lee Nodos, no considera los atributos. Una solución es utilizar el método `GetAttribute` cuando el Nodo actual sea `<libro>`:

```
private void button1_Click(object sender, System.EventArgs e)
{
      FileStream fich;

      fich   =   new   FileStream("..\\..\\libros.xml",   FileMode.Open,
      FileAccess.Read);

      XmlTextReader lector1 = new XmlTextReader(fich);

      while (lector1.Read())
      {
            if (lector1.NodeType == XmlNodeType.Text)
                  listBox1.Items.Add(lector1.Value);
            else
            {
               if    (lector1.NodeType    ==    XmlNodeType.Element    &
               lector1.Name.Equals("libro"))
                  for(int cont=0; cont<lector1.AttributeCount; cont++)
                     listBox1.Items.Add(lector1.GetAttribute(cont));
            }
      }
}
```

En caso de que el nodo actual no sea de tipo texto, se pregunta si es un elemento (`<libro>`, `<autor>`, `<titulo>`... son elementos) y si su nombre es `libro` -que es el que tiene atributos-. En tal caso se obtienen sus atributos y se añaden a la lista `listBox1`.

El resultado de ejecutar la aplicación es:

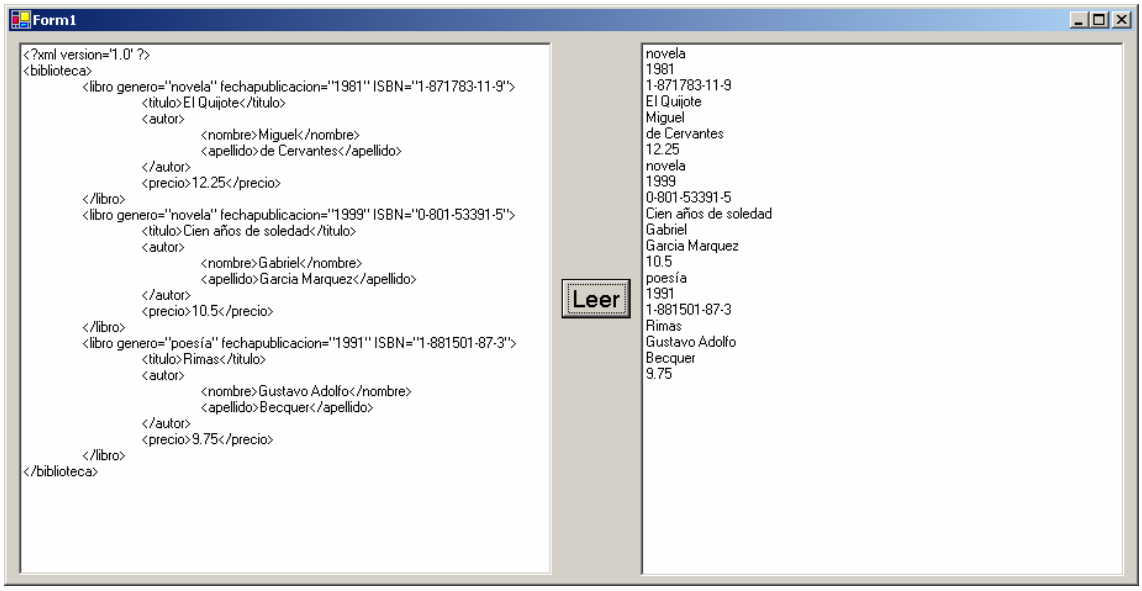


Figura 25.9. Aplicación `LecturaDocsXML1`. Utilización de las propiedades `NodeType` y `Name` y del método `GetAttribute` para mostrar también los atributos del elemento libro.

Como puede observarse es indispensable conocer bien los tipos de Nodos y cómo manejarlos para poder analizar de un modo correcto un documento XML.

Nodos XML.

Cuando un parser lee un documento XML, realmente está leyendo sus nodos uno a uno. Dependiendo del tipo de nodo en el que se esté, la información que se puede obtener del nodo es diferente y también el tratamiento que se le puede dar.

Como ya se ha visto, la propiedad `NodeType` ofrece información sobre el tipo de nodo actual al que "apunta" un `XmlTextReader` –es decir, un objeto que implemente `XmlReader`-.

`NodeType` es realmente una enumeración, cuyos miembros son:

Miembro	Descripción
`Attribute`	Un atributo. Un nodo de tipo atributo puede tener nodos hijo de tipo `Text` o de tipo `EntityReference`. Un nodo `Attribute` no aparece como nodo hijo de otro nodo, esto implica que no se considera nodo hijo de un `Element` (elemento). Ejemplo XML: `id='123'`
`CDATA`	Una sección `CDATA` es un nodo que se utiliza para poder introducir texto sin que sea parseado (de modo que no sea confundido con marcas XML). En muchos casos se utiliza para introducir código fuente (scripts...). Una nodo sección `CDATA` puede ser nodo hijo de nodos `DocumentFragment`, `EntityReference` y `Element`. Por otro lado no puede tener nodos hijo. Ejemplo XML: `<![CDATA[texto libre, aquí puede ir <<"" texto que fuera sería considerado marcas especiales]]>`
`Comment`	Es un comentario. Una nodo comentario puede ser nodo hijo de nodos `Document`, `DocumentFragment`, `EntityReference` y `Element`. Por otro lado no puede tener nodos hijo. Ejemplo XML: `<!--Este es el contenido de un nodo Comment -->`
`Document`	Es un nodo `Document` y representa al documento complete. A partir de un nodo `Document` se accede al documento completo. Puede tener como nodos hijo: - `Element` - `ProcessingInstruction` - `Comment` - `DocumentType` No puede ser nodo hijo.

`DocumentFragment`	En un fragmento de un documento. Asocia un nodo o un subárbol con un documento. Puede tener nodos hijo del tipo: - `Element` - `ProcessingInstruction` - `Comment` - `Text` - `CDATA` - `EntityReference` No puede ser nodo hijo.
`DocumentType`	Es la declaración del tipo del documento XML, se indica mediante el tag `<!DOCTYPE>`. Un nodo `DocumentType` puede tener nodos hijo del tipo `Notation` y `Entity`. Puede ser hijo de un nodo `Document`. Ejemplo XML: `<!DOCTYPE ...>`
`Element`	Es un elemento. Puede tener nodos hijo del tipo: - `Element` - `ProcessingInstruction` - `Comment` - `Text` - `CDATA` - `EntityReference` Puede ser nodo hijo de: - `Document` - `DocumentFragment` - `EntityReference` - `Element` Ejemplo XML: `<Libro>`
`EndElement`	Es un tipo de nodo que indica el final de un elemento. Es el nodo que devuelve `XmlReader` cuando se llega al final de un elemento. Ejemplo XML: `</Libro>`
`EndEntity`	Es un tipo de nodo que indica el final de un elemento (exactamente es el nodo que devuelve `XmlReader` cuando se llega al final de un elemento correspondiente a una entidad como resultado de una llamada a `ResolveEntity`.
`Entity`	Es una declaración de entidad. Puede tener como nodos hijo todos aquellos que representen la entidad expandida. Por ejemplo: `Texto`, `EntityReference`... Puede ser nodo hijo de: - `DocumentType` Ejemplo XML:`<!ENTITY MiEntidad...>`
`EntityReference`	Es una referencia a una entidad. Puede tener nodos hijo del tipo: - `Element` - `ProcessingInstruction` - `Comment` - `Text` - `CDATA` - `EntityReference`

	Puede ser nodo hijo de: - `Attribute` - `DocumentFragment` - `EntityReference` - `Element` Ejemplo XML: `&MiEntidad`
`None`	Es lo que devuelve `XmlReader` si no ha sido llamado el método `Read`.
`Notation`	Es una notación en la declaración de tipo de documento. No puede tener nodos hijo y puede ser hijo del nodo `DocumentType`. Ejemplo XML: `<!NOTATION ...>`
`ProcessingInstruction`	Es una processing instruction (`PI`). No puede tener nodos hijo y puede ser hijo de un nodo: - `Document` - `DocumentFragment` - `Element` - `EntityReference` Ejemplo XML: `<?pi test?>`
`SignificantWhitespace`	Es un espacio blanco entre marcas en un modelo de contenido mixto, o un espacio blanco con el ámbito XML: `space=”preserve”.`
`Text`	Es el texto contenido en un elemento. No puede tener nodos hijo y puede ser hijo de un nodo: - `Attribute` - `DocumentFragment` - `Element` - `EntityReference`
`Whitespace`	Es un espacio blanco entre marcas
`XmlDeclaration`	Es un nodo de tipo declaración XML . Tiene que ser el primer nodo de un documento. No puede tener nodos hijo. Es hijo del nodo raíz. Puede tener atributos que indiquen la información de versión o codificación. Ejemplo XML: `<?xml version='1.0'?>;`

Ejemplo: se va a modificar a continuación la aplicación `LecturaDocsXML1`, añadiéndole un botón `Ver Nodos` (`button2`), tal que al pulsarlo se muestre en la lista `listbox1` el tipo (`NodeType`) y nombre (`Name`) de todos los nodos del documento `libros.xml`.

El código de `button2_click` es:

```
...
...
private void button2_Click(object sender, System.EventArgs e)
{
      FileStream fich;

      fich   =   new   FileStream("..\\..\\libros.xml",   FileMode.Open,
      FileAccess.Read);
```

```
        XmlTextReader lectornodos = new XmlTextReader(fich);

        while (lectornodos.Read())
        {
              listBox1.Items.Add(lectornodos.NodeType + "--" +
                                                    lectornodos.Name);
        }

        lectornodos.Close();
        fich.Close();
}
...
...
```

El resultado de ejecutar la aplicación es:

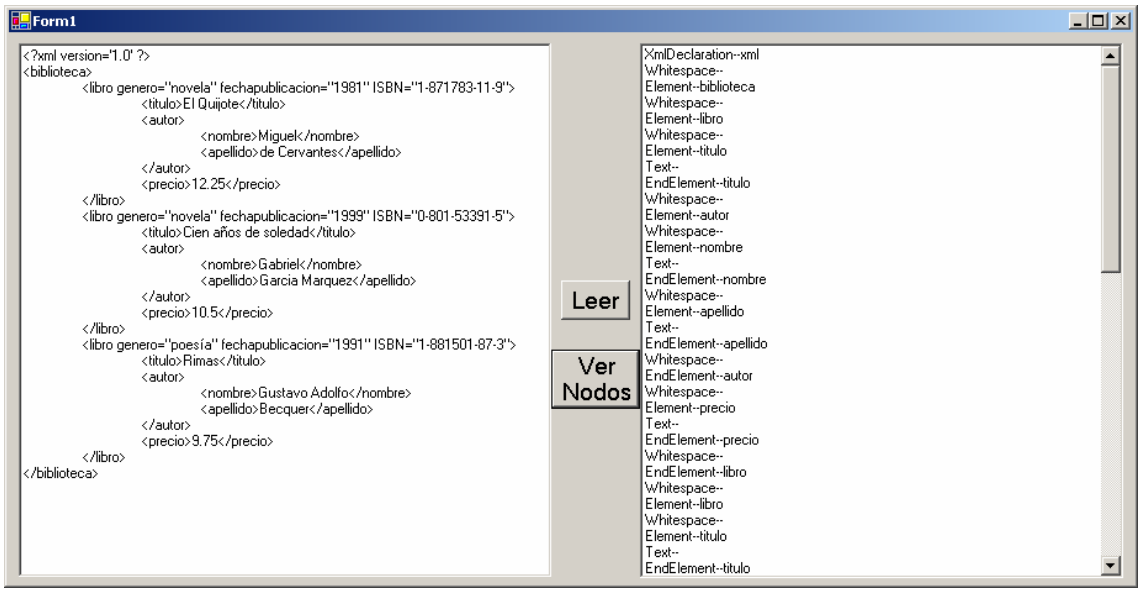


Figura 25.10. Aplicación `LecturaDocsXML1`. Utilización de las propiedades `NodeType` y `Name` para mostrar la información sobre los nodos.

Como puede observarse, los únicos nodos para los que la propiedad `Name` devuelve un valor son `XmlDeclaration`, `Element` y `EndElement`. Además, se saltan los atributos del nodo de tipo `Element <libro>` (no aparecen).

XMLNodeReader.

Esta clase permite leer un árbol DOM que contenga un documento XML.
No soporta validación. Al igual que `XmlTextReader`, en caso de querer validar el documento a leer, se ha de utilizar `XmlValidatingReader`.

La principal diferencia con `XmlTextReader` es que al constructor de `XmlNodeReader` ha de pasársele un objeto de la clase `XmlDocument`, que representa un documento cargado en memoria según el modelo DOM.

Ejemplo: se va a modificar a continuación la aplicación `LecturaDocsXML1`, añadiéndole un botón `Leer DOM` (`button3`), tal que al pulsarlo (`button3_click`) se creará un objeto

`XmlDocument` con el contenido de `libros.xml`. Se utilizará un objeto `XmlNodeReader` para leer el árbol DOM contenido por `XmlDocument`, mostrando sus nodos en la lista (completos).

El código de `button3_click` es:

```
private void button3_Click(object sender, System.EventArgs e)
{
      XmlNodeReader lectorDOM = null;

      try
      {
            //crear un objeto XmlDocument asociado a un fichero cuyo
            //nombre se pasa en nombrefich y cargar el fichero.
            XmlDocument doc = new XmlDocument();
            doc.Load("..\\..\\libros.xml");

            //crear un objeto XmlNodeReader asociado al objeto
            //XmlDocument.
            lectorDOM = new XmlNodeReader(doc);

            //Analizar el documento y mostrar sus nodos.
            while (lectorDOM.Read())
            {
                  switch (lectorDOM.NodeType)
                  {
                        case XmlNodeType.Element:
                              listBox1.Items.Add("<" + lectorDOM.Name +
                              ">");
                              break;
                        case XmlNodeType.Text:
                              listBox1.Items.Add(lectorDOM.Value);
                              break;
                        case XmlNodeType.CDATA:
                              listBox1.Items.Add(lectorDOM.Value);
                              break;
                        case XmlNodeType.ProcessingInstruction:
                              listBox1.Items.Add("<?"  +  lectorDOM.Name
                              + ", " + lectorDOM.Value + "?>");
                              break;
                        case XmlNodeType.Comment:
                              listBox1.Items.Add("<!--"              +
                              lectorDOM.Value + "-->");
                              break;
                        case XmlNodeType.XmlDeclaration:
                              listBox1.Items.Add("<?xml
                              version='1.0'?>");
                              break;
                        case XmlNodeType.Document:
                              break;
                        case XmlNodeType.EndElement:
                              listBox1.Items.Add("</"  +  lectorDOM.Name
                              + ">");
                              break;
                        }
                  }
            }
      finally
      {
            if (lectorDOM!=null)
```

```
                lectorDOM.Close();
        }

        }
}
```

Se ha utilizado una sentencia `try` para controlar el caso de que no se pueda crear el lector.

El resultado de ejecutar la aplicación es:

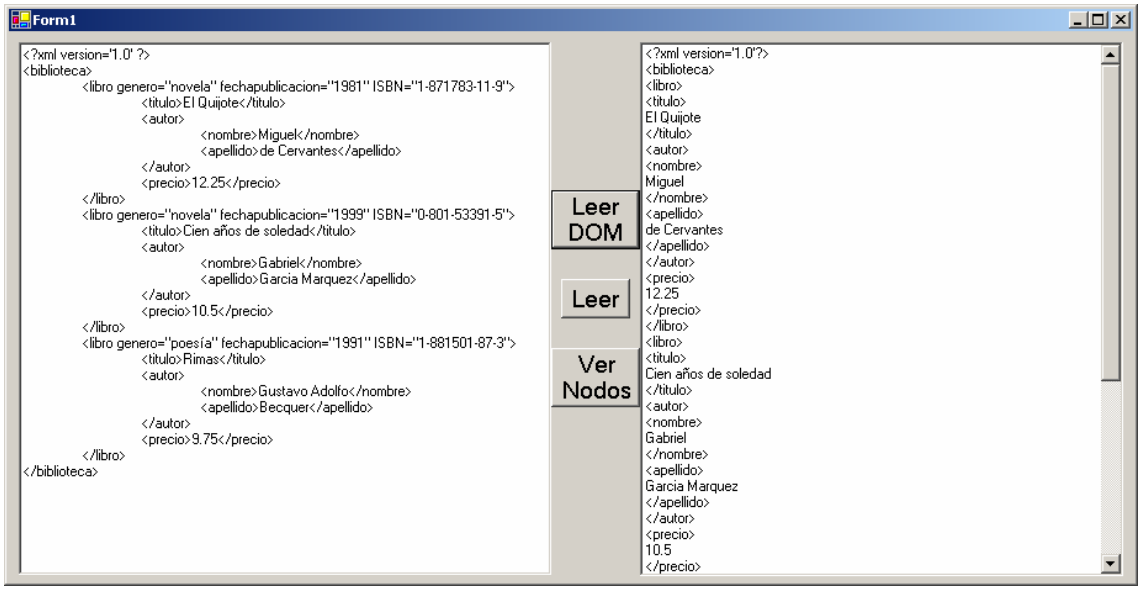


Figura 25.11. Aplicación `LecturaDocsXML1`. Utilización de `XmlDocument` y `XmlNodeReader` para mostrar la información sobre los nodos.

XmlValidatingReader.

La clase `XmlValidatingReader` es similar a `XmlTextReader`: la diferencia es que permite validar un documento XML, además de analizarlo. De hecho, una de las sobrecargas del constructor de `XmlValidatingReader` recibe como parámetro una referencia a un objeto `XmlTextReader`.

Se puede determinar si validar o no un documento XML estableciendo la propiedad `ValidationType`. Los posibles miembros de `ValidationType` son:

Miembro	Descripción
`Auto`	Si existe un DTD o un esquema (XSD o XDR) se utiliza para la validación.
`DTD`	La validación se hace según un DTD.
`None`	No se realiza validación y no se lanzan, por tanto, errores de validación.
`Schema`	Se valida según uno o varios esquemas (XSD) que han de haber sido indicados en la propiedad `Schemas`.

`XDR`	La validación se realiza según esquemas reducidos (XDR).

En caso de realizar validación, la propiedad `ValidationEventHandler` permite indicar, mediante un `delegate`, el método que manejará los eventos de validación, es decir, los errores que se den durante la validación.

Por ejemplo: se desea crear una aplicación llamada `ValidacionDocsXml` similar a `LecturaDocsXml1` que lea el documento `libros.xml` y muestre su contenido en una lista. La lectura ha de realizarse con un `XmlValidatingReader`, utilizando `libros.xsd` como `schema` para la validación.

El documento de esquema `libros.xsd` será:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="urn:biblioteca-schema"
    elementFormDefault="qualified"
    targetNamespace="urn:biblioteca-schema">

 <xsd:element name="biblioteca" type="Tipobiblioteca"/>

 <xsd:complexType name="Tipobiblioteca">
  <xsd:sequence maxOccurs="unbounded">
   <xsd:element name="libro"  type="Tipolibro"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Tipolibro">
  <xsd:sequence>
   <xsd:element name="titulo" type="xsd:string"/>
   <xsd:element name="autor" type="Tipoautor"/>
   <xsd:element name="precio"  type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="genero" type="xsd:string"/>
  <xsd:attribute name="fechapublicacion" type="xsd:string"/>
  <xsd:attribute name="ISBN" type="xsd:string"/>
 </xsd:complexType>

 <xsd:complexType name="Tipoautor">
  <xsd:sequence>
   <xsd:element name="nombre"  type="xsd:string"/>
   <xsd:element name="apellido" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

La línea **`xmlns="urn:biblioteca-schema"`** implica que el documento `libros.xml` tiene que incluirla para no tener un conflicto de nombres con `libros.xsd`. Es simplemente una mejora; recuerde los espacios de nombres en XML:

```
<?xml version='1.0' ?>
<biblioteca xmlns="urn:biblioteca-schema">
      <libro  genero="novela"  fechapublicacion="1981"  ISBN="1-871783-
      11-9">
            <titulo>El Quijote</titulo>
...
```

```
...
```

El código del método `button1_Click`, que es donde se realizará el análisis validante del documento `libros.xml`, es:

```
private void button1_Click(object sender, System.EventArgs e)
{
      FileStream fich;
      fich = new FileStream("..\\..\\libros.xml", FileMode.Open,
      FileAccess.Read);

      XmlTextReader lector1 = new XmlTextReader(fich);

      //Creación del objeto de tipo XmlValidatingReader
      //a partir del XmlTextReader
      XmlValidatingReader lector1_val= new
      XmlValidatingReader(lector1);

      //Adición de libros.xsd a la colección Schemas de lector1_val
      //libros.xsd será el esquema utilizado para validar
      lector1_val.Schemas.Add(null, "..\\..\\libros.xsd");
      //Elección de Schema como tipo de validación
      lector1_val.ValidationType = ValidationType.Schema;
      //El método manejador de los errores de validación
      //será ManejadorValidacion
      lector1_val.ValidationEventHandler += new ValidationEventHandler
      (ManejadorValidacion);

      while (lector1_val.Read())
      {
            if (lector1_val.NodeType == XmlNodeType.Text)
                  listBox1.Items.Add(lector1_val.Value);
            else
            {
              if (lector1_val.NodeType == XmlNodeType.Element &
              lector1_val.Name.Equals("libro"))
                for(int cont=0; cont<lector1_val.AttributeCount;
                cont++)
                  listBox1.Items.Add(lector1_val.GetAttribute(cont));
            }
      }
      lector1.Close();
      fich.Close();
}
```

Es importante observar que la propiedad `Schemas` es una colección a la que se añaden las rutas de los diferentes esquemas -pueden ser varios- que se desean utilizar para la validación.

La propiedad `ValidationEventHandler` indica el método manejador de los eventos que se den durante la validación, el código de este método es:

```
public static void ManejadorValidacion (object sender,
                                        ValidationEventArgs args)
{
      MessageBox.Show(args.Message);
}
```

El resultado de ejecutar la aplicación y pulsar el botón `Leer` (`button1`) es:

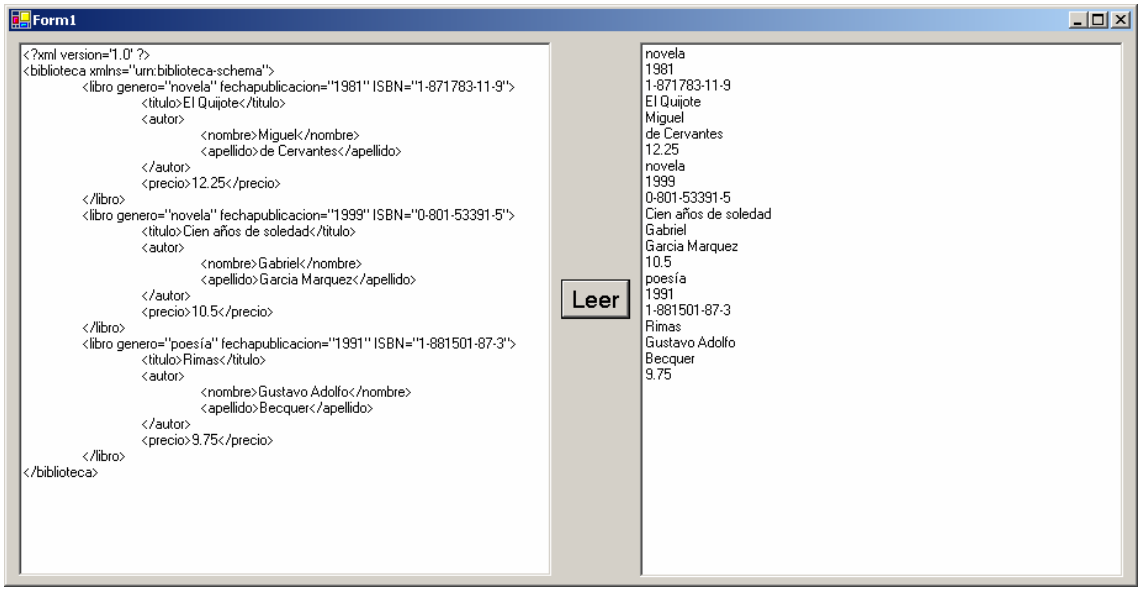


Figura 25.12. Utilización de `XmlValidatingReader` con un documento XML correcto.

Si se observa el contenido del documento `libros.xml` en la caja de texto, en la parte izquierda, puede verse el añadido

`xmlns="urn:biblioteca-schema"`.

Por lo demás, parece que todo ha ido bien y así es, ya que el documento `libros.xml` es correcto según `libros.xsd`.

Para apreciar la validación es una buena ayuda ver el resultado de un error durante la misma. Para conseguirlo, basta con hacer incorrecto el documento `libros.xml`. Por ejemplo, puede eliminarse el elemento `apellido` del primer `<libro>` en el documento `libros.xml` (`libros_error.xml`):

```
<?xml version='1.0' ?>
<biblioteca xmlns="urn:biblioteca-schema">
      <libro  genero="novela"  fechapublicacion="1981"  ISBN="1-871783-
11-9">
            <titulo>El Quijote</titulo>
            <autor>
                  <nombre>Miguel</nombre>
            </autor>
            <precio>12.25</precio>
...
...
```

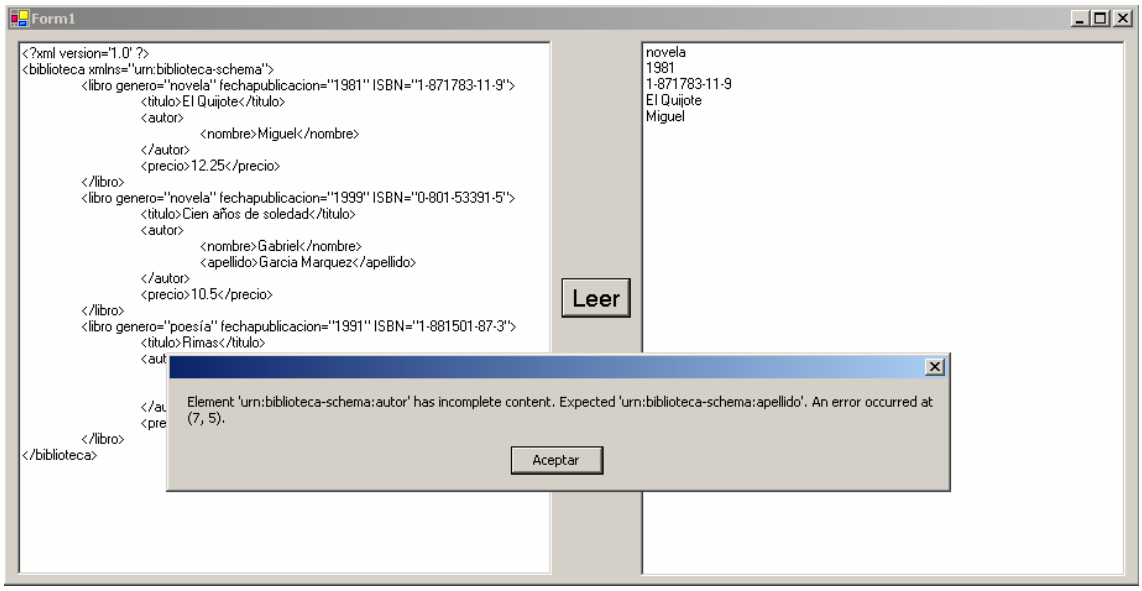


Figura 25.13. Utilización de `XmlValidatingReader` con un documento XML erróneo.

Escritura de documentos XML.

Existen dos clases derivadas de `XmlWriter`, que permiten escribir documentos XML:

- `XmlTextWriter`, que es la inversa de `XmlTextReader`. Permite escribir texto XML a un stream, a un fichero o a un objeto de la clase `TextWriter`.

- `XmlNodeWriter`, que permite escribir texto XML a un stream asociado a un árbol DOM contenido en memoria.

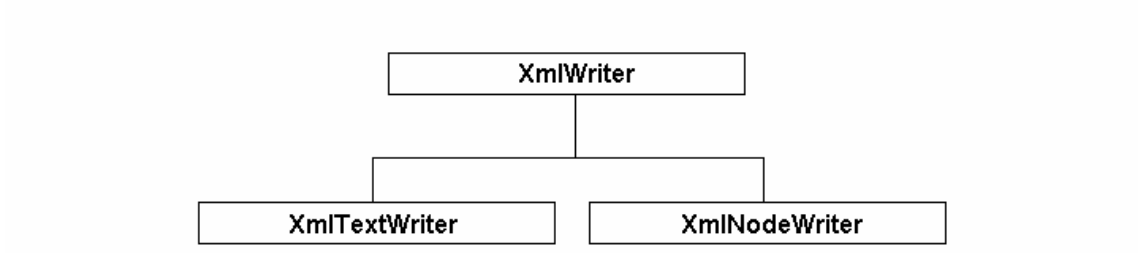


Figura 25.14. `XmlWriter` y clases que la implementan.

Los métodos más significativos de `XmlTextWriter` son los de tipo `WriteXXX`:

- `WriteStartDocument`: escribe la declaración de documento XML. Por ejemplo: `<?xml versión="1.0">`.

- `WriteEndDocument`: cierra todos los elementos y atributos abiertos y sitúa el "puntero" de escritura al principio.

- `WriteStartElement` y `WriteEndElement` permiten escribir el comienzo y el final de un elemento. `WriteEndElement` no necesita que se le pase el elemento como parámetro, escribe el que corresponda cerrar.

- `WriteElementString`: escribe un elemento que contiene un valor de tipo `string`. Equivale a llamar a `WriteStartElement`, `WriteString` y `WriteEndElement`.

- `WriteAttributes`: escribe todos los atributos que obtenga del Nodo actual en el objeto `XmlReader` cuya referencia recibe como primer parámetro.

- `WriteAttributeString`: escribe un atributo y su valor. Se deben pasar como parámetros.

- `WriteString`: escribe un `string`.

- `WriteCData`: escribe un bloque `<![CDATA[...]]>` que contiene el texto que se le pasa como parámetro.

- `WriteComment`: escribe un comentario (`<!--...-->`) con el texto que se le pasa como parámetro.

- `WriteNode`: escribe el Nodo actual del `XmlReader` que recibe como parámetro y avanza el `XmlReader` al siguiente Nodo.

XMLTextWriter.

Los métodos explicados de `XmlWriter` no están todos implementados en la propia clase `XmlWriter`, ya que es abstracta. `XmlTextWriter` implementa todos los métodos no implementados de `XmlWriter`.

`XmlTextWriter` permite escribir texto XML a un stream de modo no cacheado y escritura sólo hacia delante (`forward-only`), lo cual le da gran rapidez.

El constructor de `XmlTextWriter` está sobrecargado. Las posibles opciones son:

- `public XmlTextWriter(TextWriter)`

La escritura se hace a través de un objeto de la clase `TextWriter`, que puede estar asociado a un fichero, etc...

- `public XmlTextWriter(Stream, Encoding)`

La escritura se hace a través de un objeto de la clase `Stream`, que puede estar asociado a un fichero, etc... `Encoding` indica la codificación utilizada para el texto (`UTF-8, UTF-16`...).

- `public XmlTextWriter(Stream, Encoding)`

La escritura se hace a un fichero cuyo nombre y ruta se indica en el primer parámetro.

Por ejemplo: se desea construir una aplicación que, al pulsar un botón `Generar Xml`, genere un fichero llamado `librosgen.xml` cuyo contenido serán tres libros (igual que el fichero `libros.xml` de los ejemplos anteriores).

El código del método de respuesta al evento `Click` sobre el botón `Generar Xml` (`button1_click`) es:

```
private void button1_Click(object sender, System.EventArgs e)
{
      //Creación del objeto XmlTextWriter
      XmlTextWriter escritor =
                  new XmlTextWriter("..\\..\\librosgen.xml", null);

      //Hacer que el formato sea indentado, es decir,
      //con tabulaciones.
      escritor.Formatting = Formatting.Indented;

      //escribir el comoenzo del documento
      //<?xml version="1.0">
      escritor.WriteStartDocument();

      //comienzo del elemento <biblioteca>
      escritor.WriteStartElement("biblioteca");

      //primer libro
      //comienzo del elemento <libro>
      escritor.WriteStartElement("libro");

      //atributos del elemento <libro>
      escritor.WriteAttributeString("genero", "novela");
      escritor.WriteAttributeString("fechapublicacion", "1981");
      escritor.WriteAttributeString("ISBN", "1-871783-11-9");

      //elemento <titulo>
      escritor.WriteElementString("titulo", "El Quijote");

      //comienzo del elemento <autor>
      escritor.WriteStartElement("autor");

      //elemento <nombre>
      escritor.WriteElementString("nombre", "Miguel");

      //elemento <apellido>
      escritor.WriteElementString("apellido", "de Cervantes");

      //final del elemento <autor>
      escritor.WriteEndElement();

      //elemento <precio>
      escritor.WriteElementString("precio", "12.25");

      //final del elemento <libro>
      escritor.WriteEndElement();

      //segundo libro
      //comienzo del elemento <libro>
      escritor.WriteStartElement("libro");

      //atributos del elemento <libro>
      escritor.WriteAttributeString("genero", "novela");
      escritor.WriteAttributeString("fechapublicacion", "1999");
      escritor.WriteAttributeString("ISBN", "0-801-53391-5");

      //elemento <titulo>
      escritor.WriteElementString("titulo", "Cien años de soledad");
```

```
        //comienzo del elemento <autor>
        escritor.WriteStartElement("autor");

        //elemento <nombre>
        escritor.WriteElementString("nombre", "Gabriel");

        //elemento <apellido>
        escritor.WriteElementString("apellido", "Garcia Marquez");

        //final del elemento <autor>
        escritor.WriteEndElement();

        //elemento <precio>
        escritor.WriteElementString("precio", "10.5");

        //final del elemento <libro>
        escritor.WriteEndElement();

        //tercer libro
        //comienzo del elemento <libro>
        escritor.WriteStartElement("libro");

        //atributos del elemento <libro>
        escritor.WriteAttributeString("genero", "novela");
        escritor.WriteAttributeString("fechapublicacion", "1991");
        escritor.WriteAttributeString("ISBN", "1-881501-87-3");

        //elemento <titulo>
        escritor.WriteElementString("titulo", "Rimas");

        //comienzo del elemento <autor>
        escritor.WriteStartElement("autor");

        //elemento <nombre>
        escritor.WriteElementString("nombre", "Gustavo Adolfo");

        //elemento <apellido>
        escritor.WriteElementString("apellido", "Becquer");

        //final del elemento <autor>
        escritor.WriteEndElement();

        //elemento <precio>
        escritor.WriteElementString("precio", "9.75");

        //final del elemento <libro>
        escritor.WriteEndElement();

        //final del elemento <biblioteca>
        escritor.WriteEndElement();

        //final del documento
        escritor.WriteEndDocument();

        //vaciar el buffer del escritor
        escritor.Flush();

        //cerrar el stream
        escritor.Close();
}
```

Aunque es largo, realmente es repetitivo, pues se generan tres elementos libro, donde lo único que cambia son los datos.

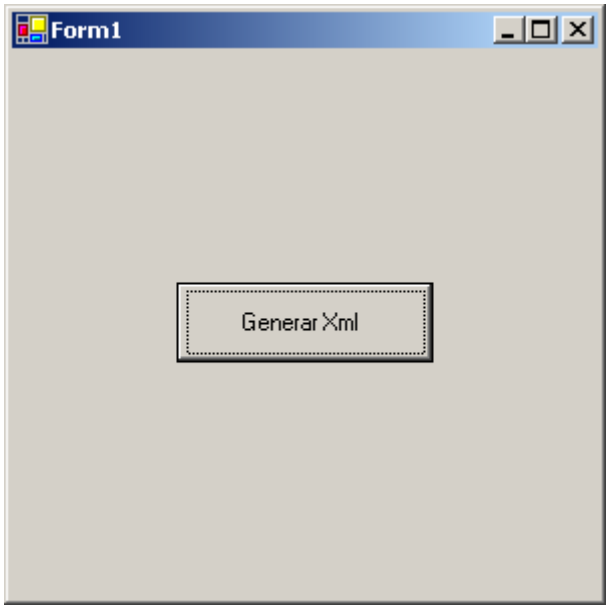
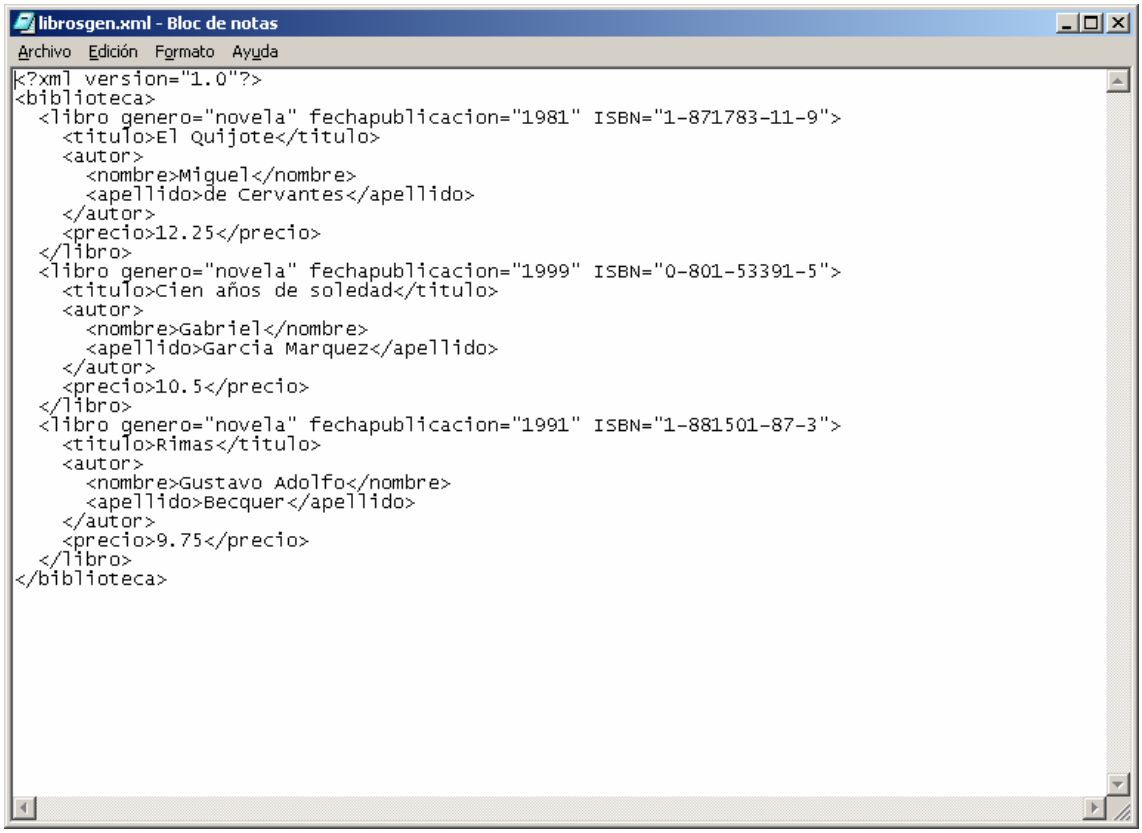


Figura 25.15. Formulario de la aplicación `GenerarDocsXml`.

Al ejecutar la aplicación y pulsar el botón `Generar Xml`, se genera, en la carpeta `GenerarDocsXml`, el fichero `librosgen.xml`.




```
<?xml version="1.0"?>
<biblioteca>
  <libro genero="novela" fechapublicacion="1981" ISBN="1-871783-11-9">
    <titulo>El Quijote</titulo>
    <autor>
      <nombre>Miguel</nombre>
      <apellido>de Cervantes</apellido>
    </autor>
    <precio>12.25</precio>
  </libro>
  <libro genero="novela" fechapublicacion="1999" ISBN="0-801-53391-5">
    <titulo>Cien años de soledad</titulo>
    <autor>
      <nombre>Gabriel</nombre>
      <apellido>Garcia Marquez</apellido>
    </autor>
    <precio>10.5</precio>
  </libro>
  <libro genero="novela" fechapublicacion="1991" ISBN="1-881501-87-3">
    <titulo>Rimas</titulo>
    <autor>
      <nombre>Gustavo Adolfo</nombre>
      <apellido>Becquer</apellido>
    </autor>
    <precio>9.75</precio>
  </libro>
</biblioteca>
```

Figura 25.16. `librosgen.xml`.

Implementación de DOM en .NET.

En el `namespace System.Xml` existen clases que implementan el modelo DOM (el nivel 1 completo y el núcleo del nivel 2).

La más importante es `XmlNode`, es una clase abstracta que representa un nodo de un documento XML. De esta clase derivan prácticamente todas las demás del modelo DOM, a excepción de `XmlNodeList`, que representa una lista o colección ordenada de nodos, `XmlImplementation`, que define el contexto para un conjunto de objetos `XmlDocument` y `XmlNamedNodeMap`, que representa una colección de nodos que pueden ser accedidos mediante un nombre o un índice.

Figura 25.17. Jerarquía de clases del modelo DOM en .NET

`XmlDocument` representa un documento XML completo. De esta clase deriva `XmlDataDocument`, que representa un documento cuyos datos pueden ser almacenados, extraídos y manejados mediante un `DataSet`. `XmlDataDocument` representa la unión de XML y ADO.

`XmlElement` representa un elemento XML y `XmlAttribute` representa un atributo dentro de un objeto de la clase `XmlElement`.

El resto de tipos de nodos XML tienen sus respectivas clases, como son `XmlNotation`, `XmlEntity`, `XmlDocumentFragment` -que representa un fragmento de un árbol DOM correspondiente a un documento- y las clases derivadas de `XmlLinkedNode`, como son `XmlDeclaration`, que representa una declaración en un documento XML (`<?xml versión='1.0' ...>`), `XmlCharacterData` -y sus derivadas, como `XmlCDataSection`...-, `XmlProcessingInstruction`, etc...

Es importante reseñar que `XmlLinkedNode` -y sus derivadas- permite obtener el nodo inmediatamente anterior -propiedad `PreviousSibling`- o posterior -propiedad `NextSibling`- al actual.

La diferencia fundamental entre utilizar el modelo DOM o las clases derivadas de `XmlReader` y `XmlWriter` es que con el modelo DOM, el documento XML se encuentra cargado completamente en memoria, pudiendo acceder a sus nodos en el orden que se quiera. En cambio, mediante las clase derivadas de `XmlReader` y `XmlWriter` no se dispone del documento XML cargado completamente, sino de un cursor asociado a un stream que sólo permite ir leyendo el documento XML asociado nodo a nodo hacia delante, sin posibilidad de acceder a nodos anteriores una vez pasados.

Esto hace ver una ventaja en el modelo DOM, que es la disponibilidad del documento completo, pero también muestra una desventaja, que es la necesidad de ocupar más memoria que simplemente leyendo el documento nodo a nodo hacia delante.

Si lo que se desea es leer o generar un documento XML directamente, en una sola pasada, es mejor utilizar clases derivadas de `XmlReader` y `XmlWriter`.

Si lo que se desea es disponer de un documento en memoria (ya sea para leer sus nodos o generarlos) pudiendo moverse entre los nodos hacia delante y atrás y modificarlos o insertar nuevos nodos, es mejor utilizar el modelo DOM.

Lectura de un documento XML utilizando el modelo DOM.

Para leer un documento XML, lo primero que se ha de hacer es crear un objeto `XmlDocument` asociado al documento y cargarlo, invocando al método `Load`.

Tras este último paso se dispondrá del árbol DOM correspondiente al documento cargado en memoria, con todos sus nodos disponibles, pudiendo utilizar los métodos de la clase `XmlDocument` para obtener los nodos. Los más importantes son:

- `FirstChild`: devuelve un objeto `XmlNode` con el nodo raíz del documento.
- `LastChild`: devuelve un objeto `XmlNode` con el último nodo.
- `ChildNodes`: devuelve una lista (`XmlNodeList`) con todos los nodos.
- `GetElementsByTagName`: devuelve una lista (`XmlNodeList`) con todos los nodos correspondientes el elemento que se le pasa como primer parámetro al método.
- Por ejemplo:

```
XmlDocument docxml = new XmlDocument();
docxml.Load("..\\..\\libros.xml");
XmlNodeList listaNodos = docxml.GetElementsByTagName("autor");
```

- `SelectSingleNode`: selecciona y devuelve el nodo cuyo nombre se le pasa como parámetro.

Todos estos métodos son heredados por `XmlDocument` de `XmlNode`, de lo cual se deduce que cualquier nodo o lista de nodos devuelta por estos métodos los soporta. Es decir, una vez se haya obtenido el nodo raíz de un objeto `XmlDocument` puede invocarse sobre el objeto `XmlNode` correspondiente al nodo raíz a cualquiera de los métodos anteriores para obtener sus nodos hijos.

A continuación se muestra mediante una sencilla aplicación de consola como utilizar `XmlDocument` y `XmlNode` para cargar un documento XML a partir de un `string` con formato XML y mostrar sus nodos por la consola.

```
using System;
using System.IO;
using System.Xml;

public class LecturaDOMDocXml
{
  public static void Main()
  {

    XmlDocument docxml = new XmlDocument();
    docxml.LoadXml("<book ISBN='1-861001-57-5'>" +
                "<title>Pride And Prejudice</title>" +
                "<price>19.95</price>" +
                "</book>");

    //El primer nodo hijo del documento es <book...>
    XmlNode nodoraiz = docxml.FirstChild;


    if (nodoraiz.HasChildNodes)
    {
      //Recorrer los nodos hijo y mostrar su contenido
      for (int i=0; i< nodoraiz.ChildNodes.Count; i++)
      {
        Console.WriteLine(nodoraiz.ChildNodes[i].InnerText);
      }
    }
  }
}
```

Este ejemplo ilustra el uso de `XmlDocument` y `XmlNode`, pero no las ventajas que ofrece el modelo DOM.

Un ejemplo más adecuado para tal fin puede ser una aplicación que ofrezca, a partir de un documento XML -por ejemplo, `libros.xml`- una lista con todos los títulos de los libros, de modo que al seleccionar uno, se muestre en una caja de texto toda su información. El hecho de implementar este ejemplo con un objeto de una clase derivada de `XmlReader` –como por ejemplo, `XmlTextReader`- obligaría a leer secuencialmente,

desde el principio, el documento `libros.xml` cada vez que se elija un título anterior al último elegido. Así la solución más acertada en este caso es utilizar el modelo DOM.

El código para cargar los títulos de los libros en la lista puede ejecutarse al cargar el formulario.

```
...
...
//creación del documento xml vacío
private XmlDocument docxml = new XmlDocument();
...
...
private void Form1_Load(object sender, System.EventArgs e)
{
      //cargar el documento en memoria
      //Se crea el árbol DOM correspondiente al documento
      docxml.Load("..\\..\\libros.xml");

      //obtención del elemento raíz del documento
      //en este caso es <biblioteca>
      XmlNode nodoraiz = docxml.DocumentElement;

      //El nodo correspondiente al elemento <biblioteca>
      //tiene 3 nodos hijos de tipo <libro>
      if (nodoraiz.HasChildNodes)
      {
            //Recorrer los nodos y añadir los títulos a la lista
            for (int i=0; i<nodoraiz.ChildNodes.Count; i++)
            {
                  //selección del nodo deseado, no es necesario
                  //pasar por los demás.
                  XmlNode nodolibro = nodoraiz.ChildNodes[i];
                  XmlNode nodotitulo = nodolibro.ChildNodes[0];
                  listBox1.Items.Add(nodotitulo.InnerText);
            }
      }
}
```

Puede observarse que se ha utilizado la propiedad `DocumentElement` para obtener el nodo raíz del documento. Esto es así porque la propiedad `FirstChild` devuelve el primer nodo del documento, que en este caso es `<?xml versión='1.0' ?>` y no el nodo raíz `<biblioteca>`, cuyos nodos hijos son los nodos `<libro>`.

El código del método de respuesta al evento `Click` sobre la lista es:

```
private void listBox1_SelectedIndexChanged(object sender,
                                                System.EventArgs e)
{
      //Selección del nodo correspondiente al libro cuyo título
      //ha sido seleccionado en la lista
      //la cadena de selección es una expresión XPath
      XmlNode nodo =
      docxml.SelectSingleNode("biblioteca/libro[titulo='" +
                              listBox1.SelectedItem.ToString() + "']");

      //mostrar los atributos, para ello se utiliza la colección
      //Attributes
```

```
        textBox1.Text = nodo.Attributes["genero"].InnerText;
        textBox2.Text = nodo.Attributes["fechapublicacion"].InnerText;
        textBox3.Text = nodo.Attributes["ISBN"].InnerText;

        //mostrar el título.
        //otro modo: nodo.SelectSingleNode("titulo").InnerText
        textBox4.Text = nodo.FirstChild.InnerText;

        //mostrar el nombre
        textBox5.Text = nodo.SelectSingleNode("autor/nombre").InnerText;

        //mostrar el apellido
        textBox6.Text                                                    =
        nodo.SelectSingleNode("autor/apellido").InnerText;

        //mostrar el precio
        textBox7.Text = nodo.SelectSingleNode("precio").InnerText;
}
```

Aunque se explica mediante comentarios es importante observar el `string` que se pasa como parámetro a `SelectSingleNode`. Es una expresión de tipo XPath que permite refinar la búsqueda del nodo deseado. El concepto es muy parecido al de un path o ruta a un fichero.

También es importante la propiedad `InnerText`, que devuelve el contenido (texto) de un nodo, ya sea dicho nodo correspondiente a un elemento, un atributo...

Una propiedad similar es `InnerXml` que devuelve el contenido XML del nodo.

Por otro lado, la colección `Attributes` contiene los distintos atributos para un nodo dado.

El resultado de ejecutar la aplicación y elegir un elemento de la lista es el de la figura 25.18.

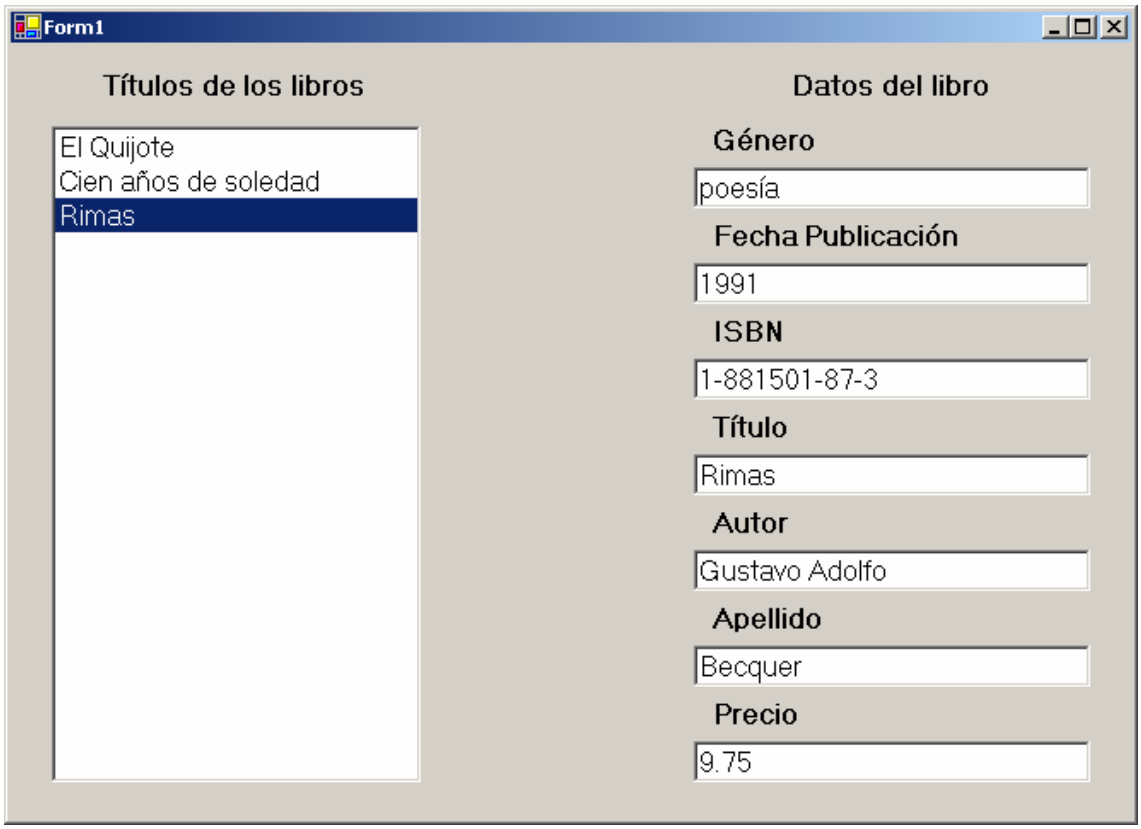


Figura 25.18. Cada vez que se selecciona un título en la lista, se accede directamente al mismo -al árbol DOM cargado en memoria-, mostrando su información en la caja de texto.

Modificación de un documento XML utilizando el modelo DOM.

Del mismo modo que es posible acceder directamente a un nodo y después obtener su contenido -`InnerText`, `ChildNodes`, `FirstChild`, `InnerXml`...- también es posible modificar su contenido.

Por ejemplo, suponga que en el ejemplo anterior se desea acceder al nodo correspondiente al libro seleccionado en la lista y modificar su atributo "`genero`" con el contenido de la caja de texto en la que se indica el género (`textBox1`), el código es:

```
//Selección del nodo correspondiente al libro cuyo título
//ha sido seleccionado en la lista
//la cadena de selección es una expresión XPath
XmlNode nodo = docxml.SelectSingleNode("biblioteca/libro[titulo='" +
                              listBox1.SelectedItem.ToString() + "']");

//actualizar el atributo genero
nodo.Attributes["genero"].InnerXml = textBox1.Text;
```

Si además se desea -utilizando el contenido de `textBox4`- modificar el título:

```
//actualizar el título en el nodo
nodolibro.SelectSingleNode("titulo").InnerText = textBox4.Text;
```

Como se puede observar, modificar el contenido de un nodo implica simplemente asignar a sus propiedades los valores deseados. No obstante, existen varios caminos diferentes para realizar operaciones sobre los nodos de un documento XML.

Aprovechando el ejemplo anterior, se va a añadir un botón `Actualizar` a la aplicación `AccesoDOMXml` que, al ser pulsado, actualice el nodo correspondiente al título seleccionado en la lista (`listBox1`) con el contenido de las cajas de texto.
El código asociado al método de respuesta al evento `Click` sobre el botón `Actualizar` será:

```
private void button1_Click(object sender, System.EventArgs e)
{
      //Selección del nodo correspondiente al libro cuyo título
      //ha sido seleccionado en la lista
      //la cadena de selección es una expresión XPath
      XmlNode                        nodolibro                          =
      docxml.SelectSingleNode("biblioteca/libro[titulo='"               +
      listBox1.SelectedItem.ToString() + "']");

      //actualizar el nodo en el árbol DOM
      this.Actualizar(nodolibro);

      //actualizar el título en la lista
      listBox1.Items[listBox1.SelectedIndex] = textBox4.Text;
}
```

El código de actualización está en el método `Actualizar`. Se ha hecho así porque es una operación que puede ser reutilizada por otras operaciones que se añadan posteriormente.

```
private void Actualizar(XmlNode nodolibro)
{
      //actualizar los atributos del nodo
      nodolibro.Attributes["genero"].InnerXml = textBox1.Text;
      nodolibro.Attributes["fechapublicacion"].InnerXml                 =
      textBox2.Text;
      nodolibro.Attributes["ISBN"].InnerXml = textBox3.Text;

      //actualizar el título en el nodo
      nodolibro.SelectSingleNode("titulo").InnerText = textBox4.Text;

      //actualizar el nombre y el resto de nodos hijo del nodo <libro>
      //seleccionado en la lista
      nodolibro.SelectSingleNode("autor/nombre").InnerText              =
      textBox5.Text;
      nodolibro.SelectSingleNode("autor/apellido").InnerText            =
      textBox6.Text;

      nodolibro.SelectSingleNode("precio").InnerText = textBox7.Text;
}
```

A continuación se muestra el resultado de ejecutar la aplicación. Al pulsar el botón `Actualizar`, el contenido de las cajas de texto es utilizado para actualizar el nodo cuyo título está seleccionado en la lista. Puede comprobarse seleccionando otro título y volviendo al correspondiente al nodo cambiado. Esta actualización se hace en el árbol

DOM en memoria, no en el fichero. Para actualizar el fichero se ha de utilizar el método `Save` de `XmlDocument`, como se explica más adelante.

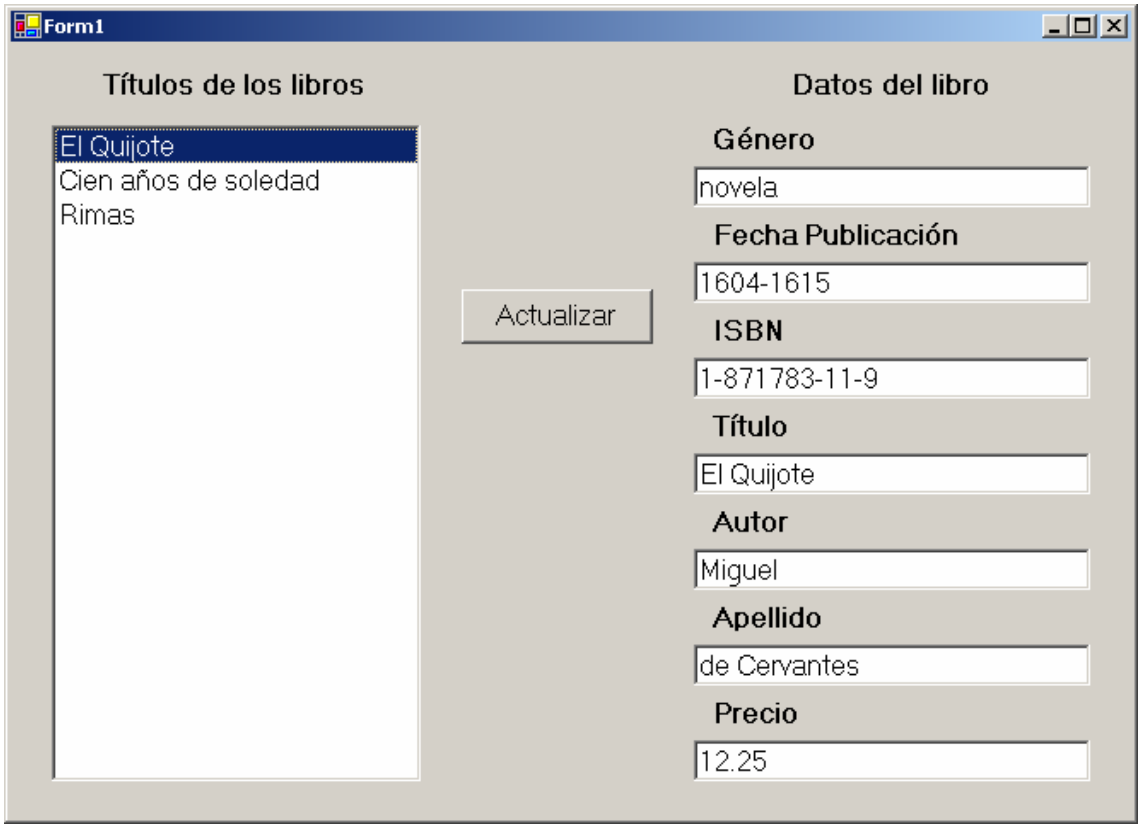


Figura 25.19. Actualización del contenido de un nodo del árbol DOM. La primera parte del Quijote se publicó en 1604 y la segunda en 1615 -la de 1614 era falsa-. Claro que podría estarse hablando de cualquier edición hasta el día de hoy. Como ejercicio pueden corregirse el resto de datos.

Generación e inserción de nuevos nodos utilizando el modelo DOM.

Antes de poder insertar un nodo en un árbol DOM se debe crear, lo cual no puede hacerse instanciando la clase `XmlNode`, porque es abstracta. Tampoco sus derivadas (`XmlElement`, `XmlAttribute`, etc...), ya que no se permite porque no se puede acceder a sus constructores.

Para crear un nodo se han de utilizar los métodos que ofrece la clase `XmlDocument`:

- `CreateNode`: este método crea un nodo, permitiendo una de sus sobrecargas indicar el tipo (`XmlNodeType`), el nombre y la URI de su `namespace`.

- `CreateElement`: es una variante de `CreateNode` que permite crear un nodo cuyo tipo es `XmlElement`.

- `CreateAttribute`: permite crear un nodo de tipo atributo.

- Además de estos tres métodos, la clase `XmlDocument` tiene otros muchos como `CreateProcessingInstruction`, `CreateCDataSection`, `CreateComment`, `CreateEntityReference`....

A continuación se muestra un ejemplo de cómo crear un nuevo nodo de tipo `Element` `<libro>` y otro de tipo `Attribute` "`genero`".

```
//creación del documento xml vacío
private     XmlDocument docxml = new XmlDocument();

//cargar el documento en memoria
//Se crea el árbol DOM correspondiente al documento
docxml.Load("..\\..\\libros.xml");

//Crear un nodo nuevo de tipo Element <libro>.
XmlNode nodolibro = docxml.CreateElement("libro");

//Creación de un nodo nuevo de tipo Attribute <genero>
XmlAttribute atributo =docxml.CreateAttribute("genero");
...
...
```

El nodo libro creado es simplemente un nodo vacío y no tiene una posición determinada en el árbol DOM referenciado por `docxml`. Para poder insertar tal nodo en el árbol DOM habrá que añadirle todos los atributos y nodos hijo que le faltan.

Otro modo de añadir un nuevo nodo a un árbol DOM ya existente es duplicar un nodo perteneciente al árbol -de igual estructura, evidentemente- utilizando el método `CloneNode` de la clase `XmlDocument`. También existe este método en la clase `XmlNode` y derivadas.

La clase `XmlNode` -y sus derivadas- permite insertar nodos en un árbol DOM mediante los métodos `InsertAfter`, `InsertBefore` y `AppendChild`.

- `InsertAfter`: recibe como parámetro un nodo y lo inserta después del nodo actual.

- `InsertBefore`: recibe como parámetro un nodo y lo inserta antes del nodo actual.

- `AppendChild`: recibe como parámetro un nodo y lo inserta después del último nodo (al final del árbol DOM).

Ejemplo. A continuación se muestra cómo insertar el nodo `nodolibro` en el árbol DOM referenciado por `docxml`.

```
//Selección del nodo correspondiente al libro cuyo título
//ha sido seleccionado en la lista
//la cadena de selección es una expresión XPath
XmlNode nodolibroactual =
            docxml.SelectSingleNode("biblioteca/libro[titulo='" +
            listBox1.SelectedItem.ToString() + "']");

//obtención del nodo raíz del dárbol DOM
XmlNode nodoraiz = docxml.DocumentElement;
```

```
//inserción del nodo nuevo en el árbol DOM
//justo detrás del actual
nodoraiz.InsertAfter(nodolibro,nodolibroactual);
```

Como puede observarse, para insertar un nodo en el árbol DOM en la posición deseada mediante `InsertAfter`, ha de utilizarse el nodo que será su padre. En este caso el raíz, `<biblioteca>`, y el nodo hermano anterior -en este caso un nodo `<libro>`-.

Si lo que se desea es insertar secuencialmente nodos hijos a partir de un nodo padre se puede utilizar el método `AppendChild` de la clase `XmlNode`, que es más cómodo.

```
//Creación de un nuevo nodo de tipo Element <titulo>
XmlElement nodotitulo = docxml.CreateElement("titulo");

//inserción del nodo nuevo en el árbol DOM, como hijo de un nodo
//<libro> (el referenciado por nodolibro)
nodolibro.AppendChild(nodotitulo);
```

Si lo que se desea insertar son atributos a un nodo, puede hacerse utilizando la colección `Attributes` del nodo, que ofrece los métodos `InsertAfter`, `InsertBefore` -similares a sus correspondientes de la clase `XmlNode`- y `Append` -no existe `AppendChild`-.

`Append` recibe como parámetro un atributo y lo añade al final de la colección `Attributes` del nodo en cuestión.

Ejemplo. A continuación se muestra cómo insertar el atributo "`genero`" en el nodo `<libro>`.

```
//Creación de un nodo nuevo de tipo Attribute <genero>
XmlAttribute atributo = docxml.CreateAttribute("genero");

// adición del atributo a la colección Attributes del nodo <libro>
nodolibro.Attributes.Append(atributo);
```

Suponga que se desea insertar un nodo con un nuevo libro a continuación del nodo actual en la aplicación `AccesoDOMDocsXml`. Para ello se añadirá un botón `Insertar`, que al ser pulsado, creará un nuevo nodo `<libro>`, tomando los datos de las cajas de texto y lo añadirá al árbol DOM, justo detrás del nodo `<libro>` actual -el seleccionado en la lista-. En la lista se insertará también el título del nuevo nodo `<libro>`. El código del método de respuesta al evento `Click` sobre el botón `Insertar` (`button2_Click`) es:

```
private void button2_Click(object sender, System.EventArgs e)
{
      //Crear un nodo nuevo de tipo Element <libro>.
      XmlNode nodolibro = docxml.CreateElement("libro");

      //Creación de un nodo nuevo de tipo Attribute <genero>
      //y adición del atributo a la colección Attributes del nodo
      //<libro>
      XmlAttribute atributo =docxml.CreateAttribute("genero");
      nodolibro.Attributes.Append(atributo);

      atributo = docxml.CreateAttribute("fechapublicacion");
      nodolibro.Attributes.Append(atributo);
```

```
        atributo = docxml.CreateAttribute("ISBN");
        nodolibro.Attributes.Append(atributo);

        //Creación y adición de todos los nodos hijos de nodolibro.
        XmlElement nodotitulo = docxml.CreateElement("titulo");
        nodolibro.AppendChild(nodotitulo);

        XmlElement nodoautor = docxml.CreateElement("autor");
        nodolibro.AppendChild(nodoautor);

        XmlElement nodonombre = docxml.CreateElement("nombre");
        nodoautor.AppendChild(nodonombre);

        XmlElement nodoapellido = docxml.CreateElement("apellido");
        nodoautor.AppendChild(nodoapellido);

        XmlElement nodoprecio = docxml.CreateElement("precio");
        nodolibro.AppendChild(nodoprecio);


        //Selección del nodo correspondiente al libro cuyo título
        //ha sido seleccionado en la lista
        //la cadena de selección es una expresión XPath
        XmlNode nodolibroactual =
              docxml.SelectSingleNode("biblioteca/libro[titulo='" +
              listBox1.SelectedItem.ToString() + "']");

        //obtención del nodo raíz del dárbol DOM
        XmlNode nodoraiz = docxml.DocumentElement;

        //inserción del nodo nuevo en el árbol DOM
        //justo detrás del actual
        nodoraiz.InsertAfter(nodolibro,nodolibroactual);

        //inserción del título en la lista
        listBox1.Items.Insert((listBox1.SelectedIndex+1),
        textBox4.Text);

        //actualizar el nodo en el árbol DOM
        this.Actualizar(nodolibro);
}
```

Se ha definido una variable referencia para cada nodo creado (`nodolibro`, `nodotitulo`, `nodoautor`, `nodonombre`, `nodoapellido`, `nodoprecio`). Ha sido así para ver de modo más claro el sentido del ejemplo pero es muy importante tener en cuenta que no es necesario crear tantas referencias -ocupan espacio en la pila-, ya que según se van creando y añadiendo nodos al nodo `<libro>` se puede reutilizar la referencia a cada nodo que ha sido añadido.

Es decir, el código

```
        //Creación y adición de todos los nodos hijos de nodolibro.
        XmlElement nodotitulo = docxml.CreateElement("titulo");
        nodolibro.AppendChild(nodotitulo);

        XmlElement nodoautor = docxml.CreateElement("autor");
        nodolibro.AppendChild(nodoautor);
```

```
XmlElement nodonombre = docxml.CreateElement("nombre");
nodoautor.AppendChild(nodonombre);

XmlElement nodoapellido = docxml.CreateElement("apellido");
nodoautor.AppendChild(nodoapellido);

XmlElement nodoprecio = docxml.CreateElement("precio");
nodolibro.AppendChild(nodoprecio);
…
…
```

puede ser sustituido por:

```
//Otro modo de crear e insertar nodos
//utilizando menos variables de referencia
XmlElement nodoaux = docxml.CreateElement("titulo");
nodolibro.AppendChild(nodoaux);

nodoaux = docxml.CreateElement("autor");
nodolibro.AppendChild(nodoaux);

XmlElement nodoaux2 = docxml.CreateElement("nombre");
nodoaux.AppendChild(nodoaux2);

nodoaux2 = docxml.CreateElement("apellido");
nodoaux.AppendChild(nodoaux2);

nodoaux = docxml.CreateElement("precio");
nodolibro.AppendChild(nodoaux);
…
…
```

Al ejecutar la aplicación aparecerá:

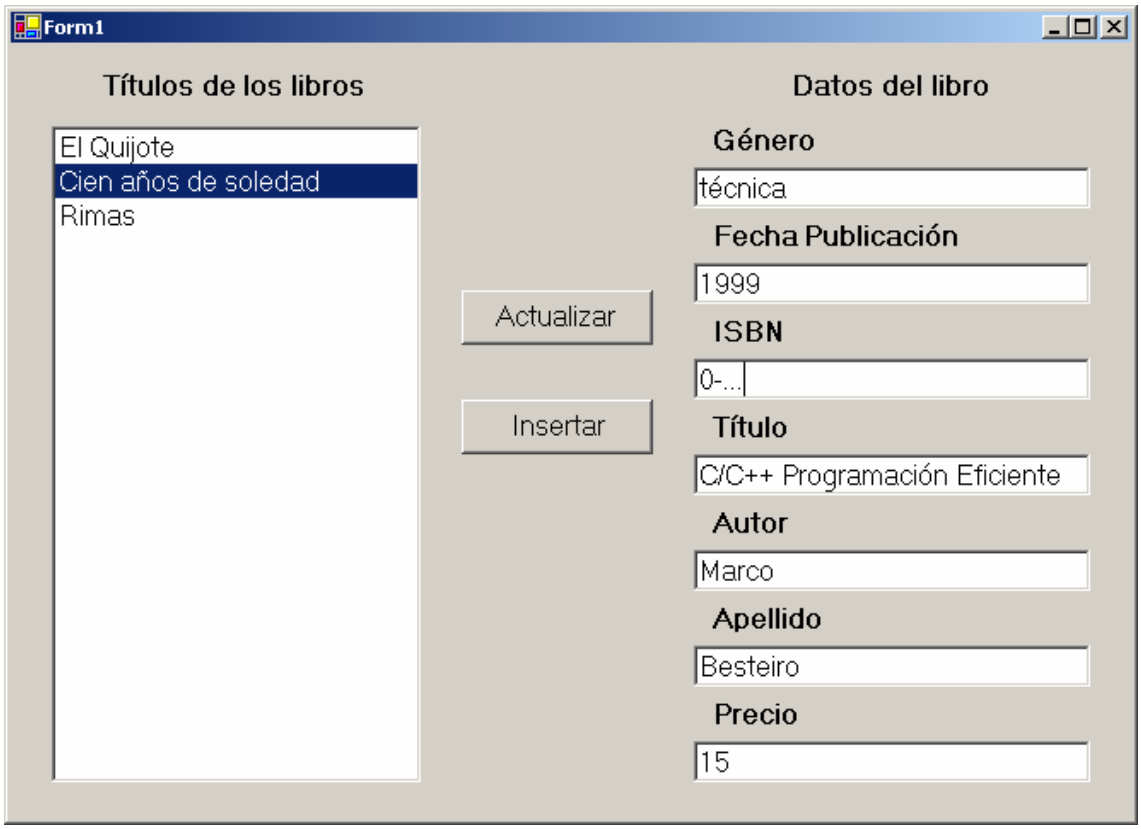


Figura 25.20. Aplicación `AccesoDOMDocsXml` al lanzarse.

Se ha seleccionado el libro cuyo título es `Cien años de soledad` y se han modificado los datos que muestran las cajas de texto. Para añadir el nuevo nodo con esos datos sólo ha de pulsarse el botón `Insertar`.

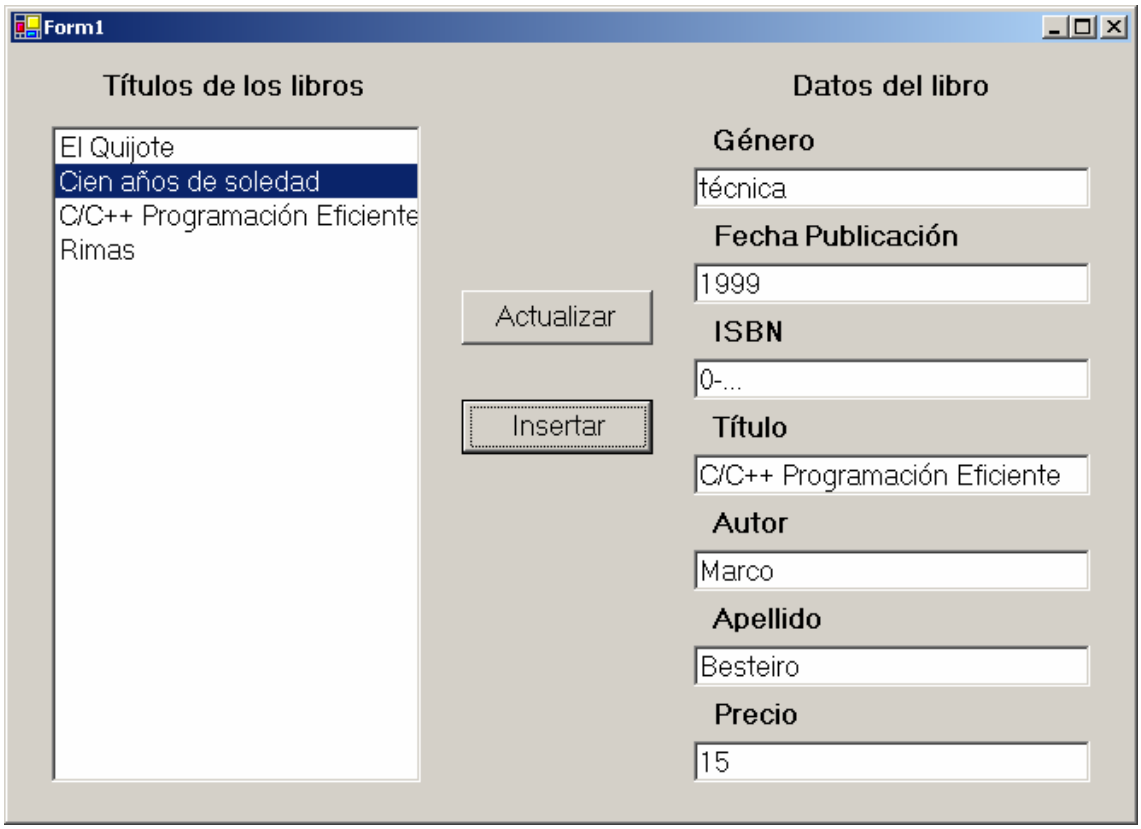


Figura 25.21. Aplicación `AccesoDOMDocsXml` tras pulsar el botón `Insertar`. El nuevo nodo libro, con los datos de las cajas texto se ha insertado tras el seleccionado en la lista.

Grabar el árbol DOM sobre un fichero XML.

Una vez que se dispone del árbol DOM en memoria y se sabe como manipularlo es importante saber cómo guardarlo en disco. Para ello, basta con invocar al método `Save` de la clase `XmlDocument` pasándole como parámetro un stream -`XmlTextWriter`, por ejemplo- asociado al fichero XML sobre el que se desea grabar el árbol DOM que contiene al documento.

Es posible utilizar el método `WriteTo`, que es similar a `Save`. La diferencia es que `Save` salva el documento en sí y `WriteTo` salva un nodo de tipo `XmlDocument`. Recuérdese que la clase `XmlDocument` deriva de `XmlNode`.

Existe otro método similar a `WriteTo` llamado `WriteContentTo`, con la diferencia de que salva todos los nodos hijo del nodo `XmlDocument` a través del cual se le invoca.

En este ejemplo se va a añadir un botón `Salvar` a la aplicación `AccesoDOMDocsXml` de modo que, al ser pulsado, el árbol DOM que se encuentre en memoria sea salvado sobre el documento `libros.xml`.

El código del método de respuesta al evento `Click` sobre el botón `Grabar` (`button3_Click`) es:

```
private void button3_Click(object sender, System.EventArgs e)
```

```
{
      XmlTextWriter            ficherolibros            =            new
      XmlTextWriter("..\\..\\libros.xml", null);

      ficherolibros.Formatting = Formatting.Indented;

      //salvar el documento cargado en memoria
      //y posiblemente modificado, sobre el
      //fichero original "libros.xml"
      docxml.Save(ficherolibros);

      //cerrar el flujo de escritura asociado al fichero
      ficherolibros.Close();
}
```

Al ejecutar la aplicación, aparecerá el botón `Grabar`, si se selecciona un libro, se insertan los datos deseados y se pulsa el botón modificar, el nuevo libro quedará añadido al árbol DOM.

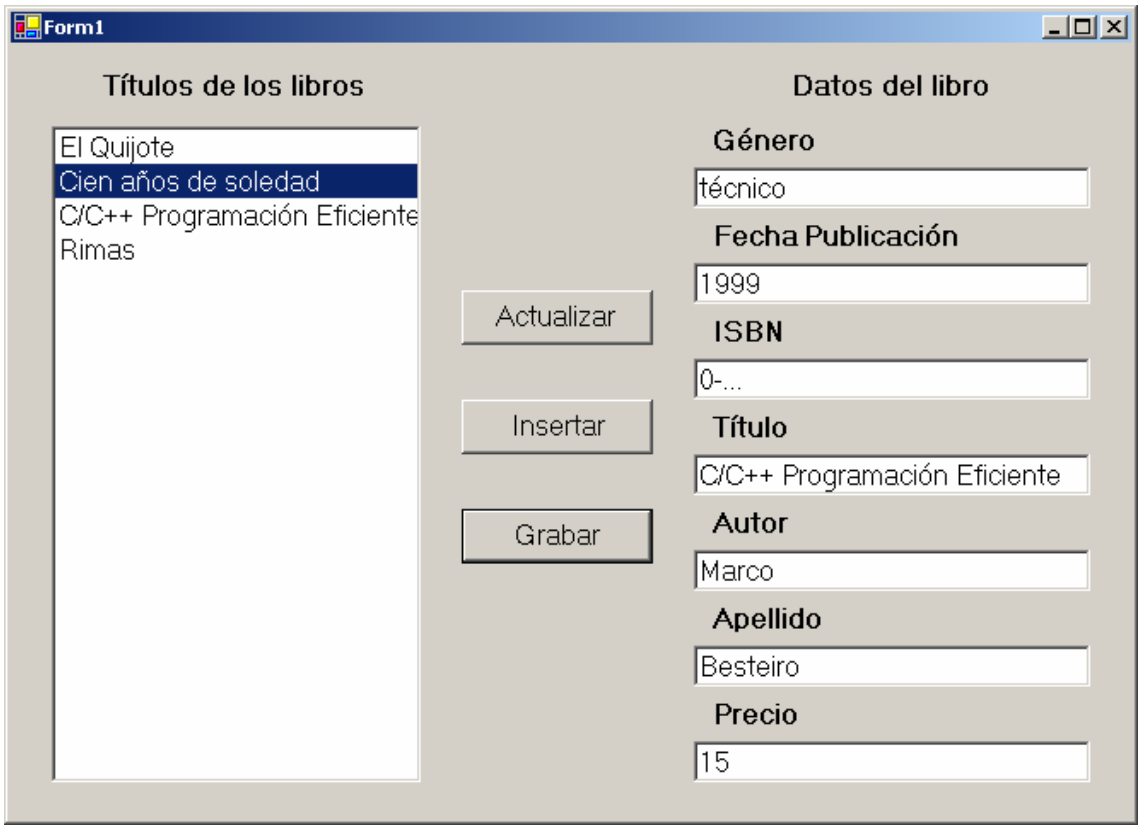


Figura 25.22. Aplicación `AccesoDOMDocsXml`. Se ha insertado el libro "C/C++ Programación Eficiente".

Al pulsar el botón `Grabar`, el fichero `libros.xml` será "machacado" con el contenido del árbol DOM.

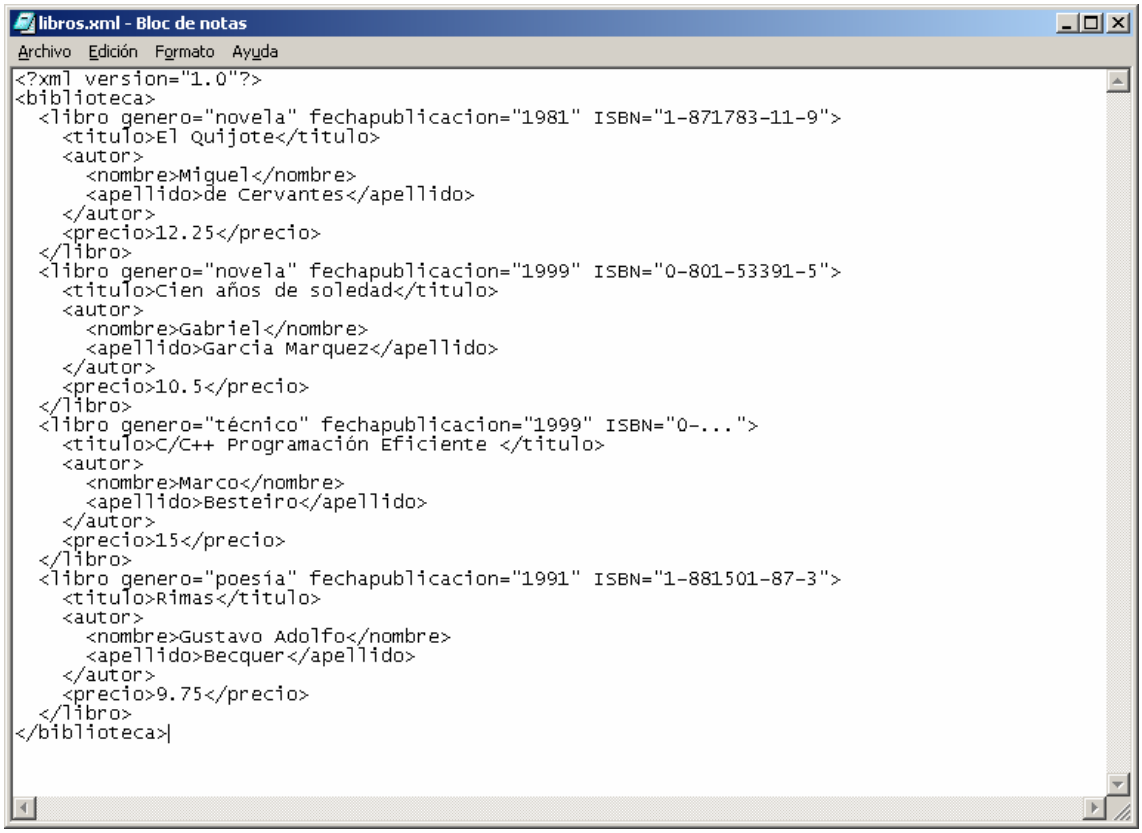


Figura 25.23. Fichero `libros.xml` con el nuevo libro.