Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

15

Web Services

Infraestructura - WSDL y UDDI

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ADO.NET avanzado

Procedimientos almacenados
SqlClient - OleDb

ISBN 978-987-1347-43-8
00015
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Practicando XML

Para ejemplificar un poco más lo expresado en las páginas anteriores, los invitamos a practicar con el siguiente ejercicio. Presentamos a continuación un Schema XML de lo que serían datos de entrenamientos de diferentes tipos de carreras:

```
<?xml version="1.0"?>
<Schema name="CarreraSchema"
  xmlns="urn:schemas-microsoft-com:xml-data"
  xmlns:dt="urn:schemas-microsoft-com:
  datatypes">
  <ElementType name="duracion" content=
  "textOnly" dt:type="time"/>

  <ElementType name="distancia" content=
  "textOnly" dt:type="float"/>

  <ElementType name="localidad" content=
  "textOnly"/>

  <ElementType name="comentarios" content=
  "textOnly"/>

  <AttributeType name="tipo" dt:type=
  "enumeration"
    dt:values="automovilística ciclismo
    natación"/>

  <AttributeType name="fecha" dt:type=
  "date"/>

  <ElementType name="sesion" content=
  "eltOnly" order="seq">
    <description>
      Este tipo de elemento representa una
      sola sesión de entrenamiento.
    </description>
    <element type="duracion" minOccurs="1"
    maxOccurs="1"/>
    <element type="distancia" minOccurs="1"
    maxOccurs="1"/>
    <element type="localidad" minOccurs="1"
    maxOccurs="1"/>
    <element type="comentarios" minOccurs="0"
    maxOccurs="1"/>
    <attribute type="tipo" default="natación"/>
    <attribute type="fecha"/>
  </ElementType>

  <ElementType name="entrenamientoLog"
  content="eltOnly">
    <description>
      Este tipo de elemento representa un
      registro de entrenamiento compuesto
      por una o mas sesiones de entrenamiento.
    </description>
    <element type="sesion" minOccurs="1"
    maxOccurs="*"/>
  </ElementType>
</Schema>
```

Con este ejercicio práctico cerramos el capítulo principal sobre XML.

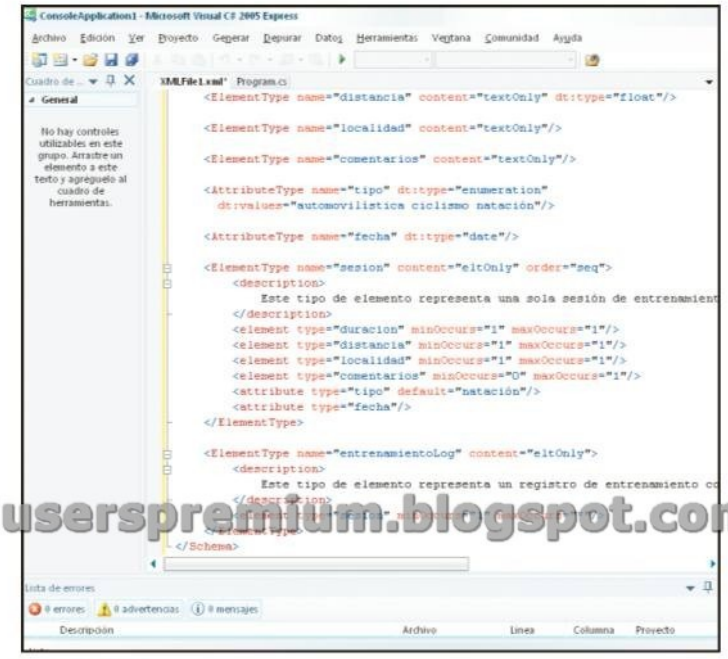


FIGURA 012 | En este caso empleamos C#, pero para realizar el ejemplo también podemos optar por hacerlo en Visual Basic .NET.

Transformar un XML

A continuación, veremos la manera de mostrar un documento XML como si fuera uno HTML.

Utilizamos XML para marcar el contenido de un documento, sin preocuparnos por el modo de mostrar la información que éste incluye. Para verlo formateado, debemos indicar a la información cómo hacerlo, especificando: fuente utilizada, colores, etc. Para lograrlo, vamos a usar las hojas de estilo (CSS).

Utilizando CSS

Una hoja de estilos CSS consta de normas que determinan cómo aplicar una serie de reglas determinadas a un documento. Estas reglas se engloban dentro de selectores que indican partes de un documento XML o HTML al cual aplicar dichas reglas; ese selector hace de vínculo entre el conjunto de estilos y el propio documento. Existen tres tipos de selectores: tipo de elemento, clase de atributo e ID de atributo.

Hojas de estilos

Podemos ampliar las hojas de estilos CSS al aplicar estilos particulares a la información que contiene un archivo XML, formateando fuentes, negrita, cursiva, etc.

A continuación, nombraremos algunas de las propiedades de estilo más comunes aceptadas por las CSS: Display, Width, Height, Border, BorderWidth, BorderColor, BorderStyle, Margin, Color, BackgroundColor, TextAlign, FontFamily, entre otras más.
Podemos crear hojas de estilo externas o internas. Aquí nos ocuparemos de las externas, ya que son las más recomendadas porque nos permiten constituir un documento aparte del que se pretende mostrar. Esto facilita el mantenimiento a la hora de querer cambiar los estilos de presentación de los datos de un XML sin modificar este último.
Para indicar qué hoja de estilo deberá utilizar un documento XML, se lo hace en el encabezado (prólogo), con la siguiente sintaxis:

```
<?xml-stylesheet type="text/css"
href="hojadeestilo.css"?>
```

Tomemos otra vez nuestro documento XML de los capítulos anteriores con la información de nuestros clientes, pero agregando la referencia a la hoja de estilos que a continuación crearemos:

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/css" href=
"clientes.css"?>
```


Tabla 5 | Cómo dar estilos a un documento XML

CSS	Cascading Style Sheets, u hojas de estilo en cascada
XSL	eXtensible Style Language, o lenguaje de estilo extensible
XSLT	XSL transformation, que deja el estilo XSL en desuso

```
<listadoClientes>
  <!–Este es un cliente–>
  <cliente>
     <apellido>de Permentier</apellido>
     <nombre>Emilio</nombre>
     <documento tipo="dni" numero="20426385">
     </documento>
     <ciudad>Resistencia</ciudad>
     <provincia>Chaco</provincia>
     <telefono>
        <prefijo>03722</prefijo>
        <fijo>123456</fijo>
        <celular>15123456</celular>
     </telefono>
  </cliente>
  <!–Este es otro cliente–>
  <cliente>
     <apellido>Raffo</apellido>
     <nombre>Fernando</nombre>
     <documento tipo="ci" numero="11444777">
     </documento>
     <ciudad>Ciudad autónoma de Buenos Aires
     </ciudad>
     <provincia>CF</provincia>
     <telefono>
        <prefijo>011</prefijo>
        <fijo>123456</fijo>
        <celular>15123456</celular>
     </telefono>
  </cliente>
</listadoClientes>
```

Veamos también un primer avance de cómo sería el código que compone nuestro archivo de hojas de estilo, al que denominamos clientes.css:

```
cliente
{
  display:block;
  width:350px;
  padding:10px;
  margin-bottom:15px;
  border:4px double red;
  background-color:Silver;
  color:Blue;
  text-align:center;
}
```

A continuación, podemos ampliar nuestra hoja de estilos aplicando otros particulares a la información que contiene; esto quiere decir que toda la información mostrada contendrá, en este caso, el estilo de cliente, salvo que se lo indique. Para ilustrar este caso, dejaremos el nombre como está, aplicaremos estilos diferentes a la dirección del cliente y no incluiremos en la muestra la información de los teléfonos. Por lo tanto, nuestro CSS quedará de la siguiente forma:

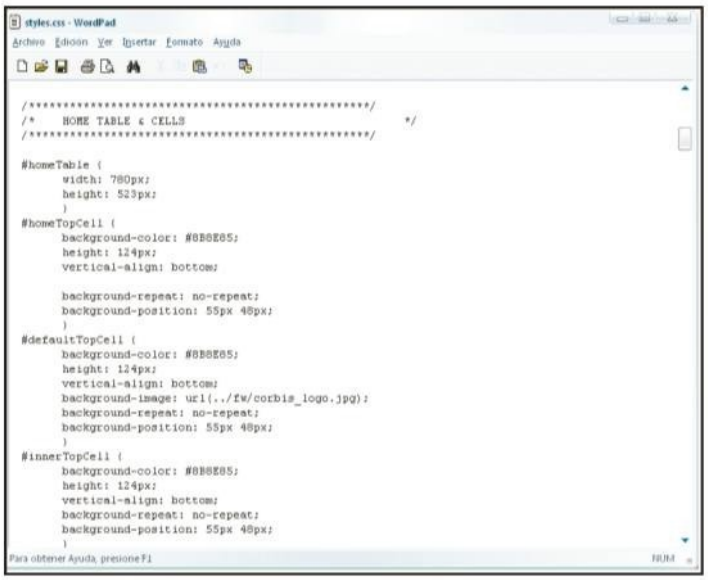


FIGURA 013 | Veamos el formato de una hoja de estilo editada directamente en WordPad.

Las hojas de estilo externas son más recomendables porque nos permiten controlar un documento aparte del que se pretende mostrar, hecho que facilita su mantenimiento.

```
cliente
{
  display:block;
  width:350px;
  padding:10px;
  margin-bottom:15px;
  border:4px double red;
  background-color:Silver;
  color:Blue;
  text-align:center;
}

ciudad, provincia
{
  color:Black;
  display:block;
  font-size:14pt;
}

telefono
{
  display:none;
}
```

XSLT

El procesador XSL lleva a cabo dos acciones fundamentales: en primera instancia, la construcción de un árbol de resultados a partir de uno de origen, y en segunda, la interpretación de dicho árbol con fines de formatear la información. Por lo tanto, con XSL existe la libertad de amoldar como se quiera el documento de origen durante la transformación, mientras que con CSS había que seguir el orden de datos del documento original XML.

Ahora presentaremos la tecnología XSLT, que es el componente de transformación de la tecnología de hojas de estilo XSL. XSLT consta de un vocabulario XML. También es posible utilizar otras tecnologías basadas en XML, como XPATH, que veremos más adelante en este curso.

Comencemos con un ejemplo. Mostraremos la directiva de procesamiento por incluir en el prólogo de nuestro documento XML:

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

Esta declaración es muy similar a la tratada con CSS. Para comprender cómo trabajar con XSL, cambiaremos primero la directiva en el archivo XML, como indicamos anteriormente, y luego presentaremos un XSL de ejemplo, que iremos explicando. El XSL es así:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/
TR/WD-xsl">
  <xsl:template match="/">
```

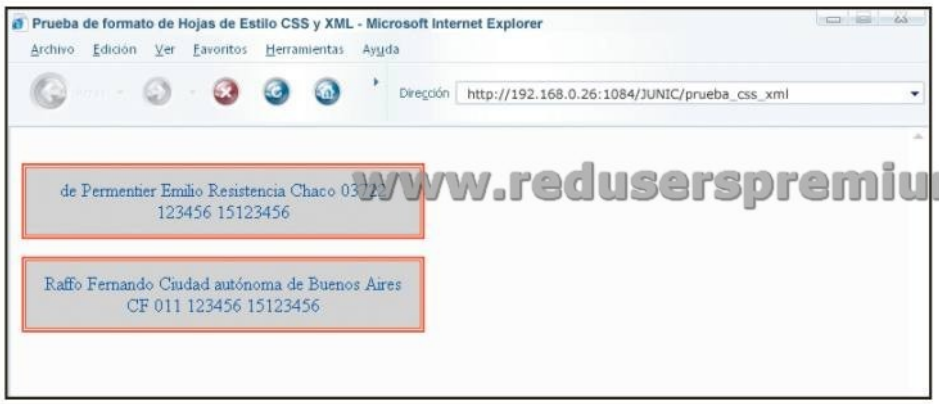


FIGURA 014 | Visualización de un XML con estilos CSS en Internet Explorer.

```
<html>
  <head>
    <title>Prueba XSLT</title>
  </head>
  <body>
    <h1 style="background-color:
    #446600; color: #FFFFFF;
    font-size: 20pt; text-align:
    center; letter-spacing:
    1.0em">Mis clientes.</h1>
    <table align="center" border="2">
      <tr>
        <th>Apellido</th>
        <th>Nombre</th>
      </tr>
      <xsl:for-each order-by="+
      apellido" select="listado
      Clientes/cliente">
        <tr>
          <td>
            <xsl:value-of
            select=
            "apellido" />
          </td>
          <td>
            <xsl:value-of
            select=
            "nombre" />
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Obtendremos un resultado en forma de tabla, con un título y un nombre que aparecerán en la barra de título del explorador.
La principal diferencia entre lo visto de CSS y lo que se explicará acerca de XSL y XSLT es que XSL se escribe en formato XML.
Si no tenemos presentes los conocimientos aprendidos en fascículos anteriores de HTML, recomendamos darle un repaso al tema, para entender que todas las etiquetas que no anteponen el prefijo XSL son propias de HTML. Esto es realmente así, con la salvedad de los datos, que serán extraídos de nuestro XML como veremos a continuación. En el tag <table> vemos cómo se crea una tabla propia de HTML donde colocaremos un título a las columnas (apellido y nombre). Pero a continuación vemos una etiqueta que hace mención a un elemento propio de XSL, en este caso, un foreach. Esta instrucción recorre todos los elementos que estén dentro del documento en **listadoClientes/cliente**, con el agregado de que los ordenará en forma ascendente (+) según el apellido. Luego, vemos dentro de este bucle cómo se "selecciona" el valor que se

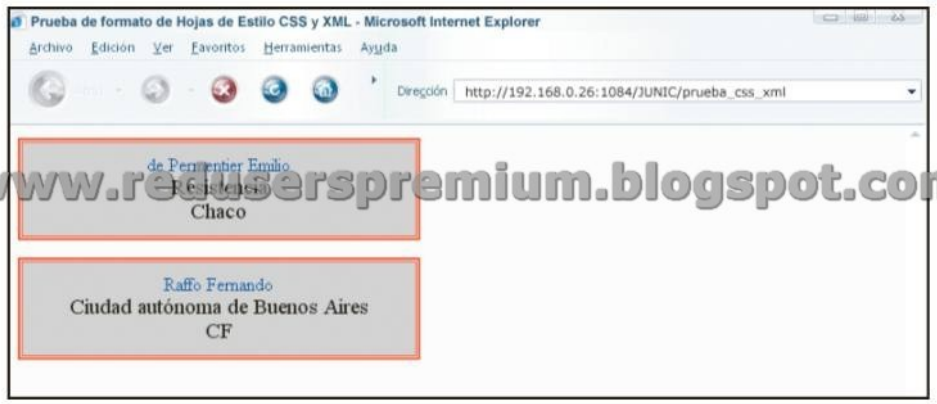


FIGURA 015 | Visualización del mismo XML con más estilos aplicados mediante CSS en Internet Explorer.

mostrará en cada celda de la tabla. En la Tabla 6 vemos las instrucciones más comunes de este tipo propias de XSL. Un ejemplo del elemento XSL if es el siguiente:

```
<xsl:if apellido="de Permentier">
  <xsl:apply-templates select
  "apellidoEspecial" />
</xsl:if>
```

Y uno del elemento XSL foreach:

```
<xsl:for-each select="listadoClientes/cliente">
  <xsl:sort select="apellido">
     <tr>
        <td>
           <xsl:value-of select="apellido" />
        </td>
        <td>
           <xsl:value-of select="nombre" />
        </td>
     </tr>
  </xsl:sort>
</xsl:for-each>
```

El resultado es el mismo, pero separamos foreach de order, para conocer un nuevo elemento: **xsl:sort**. Un ejemplo del elemento XSL choose es el que sigue:

```
<xsl:choose>
     <xsl:when test="expression">
        ... alguna salida ...
     </xsl:when>
     <xsl:otherwise>
        ... otra salida ....
     </xsl:otherwise>
</xsl:choose>
```

Hasta aquí han quedado demostradas las inmensas posibilidades que brindan los documentos XML y su asociación con diferentes tecnologías, como XSL, que también implementan formato XML. Existen formas de capturar y, por ende, de trabajar con parte de un documento XML (tecnología XPath), que desarrollaremos en los próximos capítulos. Lo que queremos expresar aquí es que, a partir de obtener "un fragmento" de un documento XML, se podría dar un estilo o formato diferente a esa parte sin afectar el resto del documento mencionado.

FIGURA 016 | El mismo XML con estilos aplicados según un XSL.

Tabla 6 | Elementos de XSL

Elemento XSL	Descripción
Valueof	Se utiliza para indicar el valor que debe colocarse en ese momento dentro del resultado de la transformación.
If	Se usa para cotejar condicionalmente.
foreach	Se emplea para recorrer un subconjunto de datos.
applytemplates	Su uso está destinado a aplicar plantillas que se definen en una hoja de estilos. Soporta los atributos order-by y select, al igual que foreach. Cuando el procesador encargado de la transformación encuentra este elemento, aplica la plantilla cuyo nombre está asociado al atributo select.
choose	Permite expresar múltiples condiciones.

Introducción a XPath

XPath es un lenguaje de expresión utilizado para extraer una parte de un documento XML, sin implementar XML. Veamos un poco más.

El uso de esta tecnología está reservado a situaciones especiales en las que el marcado XML no se aplica en realidad, como en los valores de los atributos. Al igual que XSLT, XPath supone que un documento XML ha sido analizado sintácticamente en un árbol de nodos, es decir que está correctamente formado. Siempre hay un nodo raíz que servirá como raíz de un árbol XPath y que, lógicamente, aparecerá como primer nodo del árbol en cuestión. Dentro de un nodo de elementos hay otros tipos de nodos que corresponden a su contenido. Los nodos de elementos pueden contener un identificador único que será el que se utilice para hacer referencia al nodo desde XPath. Hay varios tipos de nodos que pueden aparecer en un árbol XPath: nodos raíz, de elementos, de texto, de atributos, de espacio de nombres, de instrucciones de procesamiento y de comentarios. Ahora bien, a la hora de buscar un nodo determinado, XPath recorre el árbol de nodos del documento XML, y este recorrido se lleva a cabo mediante expresiones, cada una de las cuales, al ser evaluada, acaba siendo un objeto de datos de uno de los siguientes tipos: node-set (conjunto de nodos), boolean (true o false), número flotante o string. Cuando hablamos de nodo raíz, no debemos confundirlo con el elemento raíz del documento XML. XPath no es un lenguaje como C o Visual Basic, sino que es del tipo declarativo. Por lo tanto, las instrucciones abarcan cierta variedad de operaciones, por ejemplo: llamadas a funciones y **location paths**, que puede explicarse como caminos de localización (de elementos).

```
<?xml version="1.0" encoding="utf-8" ?>
<listadoClientes>
  <!—Este es un cliente—>
  <cliente>
     <apellido>de Permentier</apellido>
     <nombre>Emilio</nombre>
     <documento tipo="dni" numero="20426385">
     </documento>
     <ciudad>Resistencia</ciudad>
     <provincia>Chaco</provincia>
     <telefono>
        <prefijo>03722</prefijo>
        <fijo>123456</fijo>
        <celular>15123456</celular>
     </telefono>
  </cliente>
  <!—Este es otro cliente—>
  <cliente>
     <apellido>Raffo</apellido>
     <nombre>Fernando</nombre>
     <documento tipo="ci" numero="11444777">
     </documento>
     <ciudad>Ciudad autónoma de Buenos Aires
     </ciudad>
     <provincia>CF</provincia>
     <telefono>
        <prefijo>011</prefijo>
        <fijo>123456</fijo>
        <celular>15123456</celular>
     </telefono>
  </cliente>
</listadoClientes>
```

Sobre la base de este documento XML comenzaremos a ver cómo se trabaja con XPath.

La manera en que XPath se refiere a los trayectos citados en el ejemplo anterior es similar a como lo haríamos nosotros para hallar una carpeta o un archivo dentro del disco duro. Por ejemplo, la siguiente expresión en

XPath: /listadoClientes/cliente hace referencia a todos los clientes que contenga nuestro archivo XML. Esto es bastante útil al momento de tener que recorrer los clientes; pero si, en cambio, nuestra expresión fuera /listadoClientes/cliente/telefono, haría referencia a cada uno de los teléfonos de cada uno de ellos. Ahora bien, ¿qué haríamos al conocer un listado de teléfonos sin saber a quién pertenece?

Veamos entonces ciertas propiedades o referencias que posee cada nodo. Cada uno sabe quién es su padre, parentNode, por lo que, al seleccionar un teléfono en particular y haciendo referencia a su padre, cliente, luego podríamos conocer sus datos. También existen propiedades como chilNodes, que no es más que una colección de nodos hijos, o sea, los que "cuelgan" del nodo en cuestión. Cabe aclarar que el nodo raíz es el único que carece de nodo padre. Hasta ahora vemos que XPath es como una forma de obtener una colección de nodos. Pero adentrémonos un poco más para poder buscar uno con determinadas características. Por ejemplo, si queremos obtener los nodos documento, la expresión XPath sería: /listadoClientes/cliente/documento, pero esto nos traería todos los documentos que encontrara. En nuestro XML serían dos. Si quisiéramos obtener uno de ellos, podríamos utilizar la siguiente expresión: /listadoClientes/cliente/documento[@numero="20426385"]. Este tipo de expresiones se utiliza, por ejemplo, en documentos en los que se aplica la tecnología XLST, como cuando se hace un select. A continuación, veremos expresiones para seleccionar diversos elementos:

Seleccionar todos los clientes:

```
/listadoClientes/cliente
```

Seleccionar atributos número de documento:

```
/listadoClientes/cliente/documento/@numero
```

Ejemplo de un árbol de nodo

FIGURA 017 | Visualización del árbol del documento XML anterior.

En caso de que un atributo sea de carácter opcional, podríamos seleccionar los elementos que contengan al atributo número de documento:

```
/listadoClientes/cliente/documento[@numero]/*
```

Y también contamos con la posibilidad de seleccionar todos los documentos que sean DNI:

```
/listadoClientes/cliente/documento[@tipo="dni"]
```

Con una doble barra (//) podemos seleccionar de manera más simple todos los apellidos:

```
/listadoClientes//apellido
```

O los elementos que desciendan de cliente:

```
/listadoCliente/cliente//
```

Por otro lado, podríamos obtener todos los clientes que tienen al menos un teléfono, para el caso de que este dato no sea de carácter obligatorio. O sea, obtendremos todos los nodos cliente que tengan al menos un hijo del tipo telefono:

```
/listadoClientes/clientes/telefono/..
```

También existe una variedad de funciones para obtener, por ejemplo, el texto (directamente el texto) de los apellidos (cabe destacar que sin text() obtendríamos el nodo, no el texto):

```
/listadoClientes/clientes/apellido/text()
```

Hay otras funciones, como comment(), para el caso de querer obtener un comentario; o processing-instruction(), para las órdenes de procesamiento de los documentos XML.

Jerarquía de los objetos pertenecientes a la colección System y System.XML

- System.Object
  - System.Exception
    - System.System.Exception
      - System.Xml.Xpath.XPathException
  - System.ValueType
    - System.Enum
      - System.Xml.Xpath.XmlCaseOrder
      - System.Xml.Xpath.XmlDataType
      - System.Xml.Xpath.XmlSortOrder
      - System.Xml.Xpath.XpathNamespaceScope
      - System.Xml.Xpath.XpathNodeType
      - System.Xml.Xpath.XpathResultType
  - System.Xml.Xpath.XpathDocument
  - System.Xml.Xpath.XpathExpression
  - System.Xml.Xpath.XpathNavigator
  - System.Xml.Xpath.XpathNodeIterator

FIGURA 018 | Jerarquía para el espacio de nombres System.Xml.XPath.

Veamos cómo se utilizan estas expresiones en C#. Para comenzar, tenemos dos líneas:

```
XmlNode root = miDocumento.DocumentElement;
XmlNodeList lista = root.SelectNodes
("cliente");
```

En este caso, root contiene el elemento raíz de nuestro documento XML, o sea, el denominado listaClientes. Luego le pedimos que seleccione los nodos de tipo cliente, por lo que lista contendrá una colección de dos nodos para nuestro ejemplo. Si quisiéramos obtener un cliente en particular, podríamos hacerlo así:

```
lista = root.SelectNodes("cliente
[apellido='de Permentier']");
```

Igualmente, obtendríamos una colección de nodos, pero en este caso con un único elemento. Para recorrer la colección e ir trabajando con cada uno de los nodos obtenidos, se recurre a una instrucción foreach:

```
foreach (XmlNode miNodo in lista)
{
    // Trabajo con mi nodo.
}
```

Existe un espacio de nombres denominado System.Xml.XPath. Estamos en condiciones de realizar una aplicación en C# que nos permita listar los clientes que residan en el Chaco:

```
XPathNavigator nav;
XPathDocument doc;
XPathNodeIterator nodeIter;
String strExpresion;
// Apertura del documento XML mediante XPath.
doc = new XPathDocument(@"c:\archivo xml1.
xml");
// Crea un navegador de SOLO LECTURA para
consultas XPath.
nav = doc.CreateNavigator();
// Obtención de los clientes que contienen
tipo de documento DNI.
strExpresion = "/listadoClientes/cliente/
apellido[../provincia='Chaco']";
// Selección de los nodos que correspondan a
la consulta.
nodeIter = nav.Select(strExpresion);
Console.WriteLine("Lista de clientes del
Chaco:");
//Recorremos la lista en el orden en que vino.
while (nodeIter.MoveNext())
{
  Console.WriteLine("Apellido: {0}", nodeIter.
  Current.Value);
};
// Pausa.
Console.ReadLine();
```

Queda para el lector seguir investigando, ya que el universo de posibilidades es muy extenso.

Tabla 7 | Clases de System.Xml.XPath

Clase	Descripción
XPathDocument	Brinda un objeto de sólo lectura para el procesamiento de documentos XML mediante XSLT.
XPathException	Excepción que se genera a partir de un error al procesar una expresión XPath.
XPathExpression	Encapsulamiento de una expresión XPath compilada. Esta clase es utilizada por los métodos Select, Evaluate y Matches.
XPathNavigator	Lee datos de un almacén, como lo haría cualquier cursor.
XPathNodeIterator	Proporciona un iterador para el conjunto de nodos seleccionados.

ADO.NET Avanzado

Programación en acceso a base de datos

8

Contenidos

Aprendamos un poco más sobre el contenido de las clases de acceso a datos que nos brinda ADO.NET, para acceder a los distintos tipos de bases de datos existentes en el mercado.

Temas tratados

- Los espacios de nombre System.Data
- Los proveedores de nombre propios y externos
- Cadenas de conexión
- Procedimientos almacenados
- OleDB y seguridad en el acceso a SQL Server

ADO.NET Avanzado

En este capítulo, realizaremos un recorrido por los tipos de conexiones a datos como OleDB, a través de las clases brindadas por el namespace System.Data.

Introducción

Repasaremos el namespace System.Data, y todas las propiedades y métodos que brinda.

- Las capas de acceso System.Data.Common
- Proveedores de acceso a datos
- DataSet
- Conexión OleDB

Los proveedores de nombre propios y externos

La clase de acceso DataSet para manejar colecciones de tablas en memoria.

- System.Data.SQLClient
- El objeto de colección DataTable
- DataType para enumerar los tipos de datos
- DataView para el modelo transaccional

Cadenas de conexión

Veamos las clases necesarias para el modelo de conexión con motores SQL Server.

- SqlAdapter y SqlConnection
- Crear y Ejecutar comandos con sqlCommand
- El objeto SqlDataReader
- System.Data.OleDB, acceso a datos genérico

Procedimientos almacenados

Cómo utilizar una lógica para escribir procedimientos almacenados y usarlos mediante parámetros.

- Llamar a un procedimiento almacenado
- Creación de parámetros
- Utilización del objeto SqlParameter
- Pautas de su uso en VB.NET y C#

OleDB y seguridad en el acceso a SQL Server

Cómo aprovechar la tecnología OleDB que sigue vigente con el correr del tiempo, y cómo no descuidar la seguridad en las bases de datos.

- Los objetos System.Data.OleDB
- La clase OleDBParameter
- Conectándonos mediante OleDB
- Seguridad en bases de datos mediante usuarios

ADO.NET Avanzado

En el siguiente capítulo, haremos una reseña de los objetos fundamentales de ADO.NET y su uso.

Espacios de nombres

Dentro de la categoría System.Data encontramos los namespaces que nombraremos a continuación. En ellos podemos apreciar clases similares entre sí para acceder y manipular datos en memoria.

System.Data

Dentro del namespace principal, hay diversas clases genéricas para almacenamiento de datos en memoria. Entre las más relevantes encontramos: DataSet, DataTable, DataRow, DataColumn y DataType. Más adelante analizaremos en detalle la clase System.Data.

System.Data.Common

Esta categoría nos permite crear capas de acceso a datos en forma independiente del motor, con métodos, clases y objetos comunes a cualquier proveedor de base de datos dentro del framework .NET.
Las clases principales, también implementadas para cada motor, se dividen en:

- DataAdapter: Permite definir comandos de selección, inserción y actualización, para efectuar dichas operaciones desde y hacia un conjunto de datos DataSet.
- DbConnection: Representa una conexión a un motor de base de datos. Se basa, principalmente, en la cadena de conexión.
- DbParameter: Permite el envío de parámetros del tipo entrada/salida hacia el motor de base de datos, especialmente, a funciones o procedimientos almacenados.
- DbDataReader: Si sólo necesitamos leer datos de una fuente u origen, esta clase permite una rápida lectura "sólo-hacia-adelante".

> System.Data contiene todas las clases necesarias para acceso, manipulación, persistencia y actualización de datos.

Proveedores de acceso a datos

El framework .NET cuenta con clases específicas para cada uno de los motores de base de datos utilizados en la actualidad. Poseen toda la lógica necesaria al momento de conectarse al motor, ejecutar consultas y obtener resultados: SQLClient, para SQLServer; OleDBClient para conexiones OleDB; OracleClient para motores Oracle, y así sucesivamente.

DataSet

Para comprender el manejo de datos en memoria, es preciso entender el funcionamiento del objeto **Principal**, destinado a la representación en objetos de datos relacionales.

Vistazo general

Cada objeto DataSet puede contener una o más tablas, objetos DataTable y las correspondientes relaciones (conocidas como Relations).

Una vez creada la o las tablas en nuestro DataSet, será preciso indicar qué columnas poseerá, agregando instancias de la clase DataColumn.

Creación de un DataSet

El constructor de la clase DataSet tiene un parámetro de tipo String que nos sirve para especificar su nombre:

```
'Codigo VB.NET
Dim ClientesDS As DataSet = New DataSet
("Clientes ")

//Codigo C#
DataSet ClientesDS = new DataSet("Clientes ");
```

Agregar una tabla

Para contener datos, es necesario añadir DataTables a nuestro DataSet. En el siguiente fragmento vemos cómo realizar este proceso:

```
'Codigo VB.NET
Dim ClienteDT as DataTable = ClientesDS.
Tables.Add("TablaClientes")
//Codigo C#
DataTable ClienteDT = ClientesDS.Tables.
Add("TablaClientes");
```

Añadir columnas a la tabla

Una vez creada la o las tablas en nuestro DataSet, será preciso indicar qué columnas poseerá, agregando instancias de la clase **DataColumn**, como vemos a continuación.

Cargando esquemas

La clase DataSet es capaz de generar automáticamente tablas, columnas, relaciones y demás, a partir de un archivo de Esquema XML (XML Schema) con extensión XSD.
Para esto utilizaremos el método **ReadXMLSchema**.

FIGURA 001 | En www.microsoft.com/spanish/msdn/centro_recursos/ado/menu/avanzado.mspx encontraremos la información más reciente sobre ADO.NET.

```
'VB.NET
Dim DSPrueba As DataSet = New DataSet
DSPrueba.ReadXmlSchema("MiSchema.xsd")
```

Infiriendo el esquema

Si tenemos nuestros datos crudos, en formato XML, el DataSet puede, a su vez, inferir el esquema subyacente de ellos con el método **InferXMLSchema**:

```
'Siguiendo con el ejemplo anterior
DSPrueba.InferXMLSchema("ArchivoXML.xml")
```

Serializar

La forma más rápida que tenemos aquí para almacenar datos en memoria directamente en disco es utilizando la capacidad de serialización de la clase DataSet.
El método WriteXML se encarga de guardar automáticamente todo el DataSet, con sus correspondientes datos, e indica sólo la ruta del archivo de destino.

DataSets tipados

Para diseñar un DataSet de forma Visual, tenemos que agregar un nuevo elemento de tipo DataSet a nuestro proyecto actual, a través del menú Agregar nuevo elemento…, en la opción correspondiente, como se ve en la Figura 2. Acto seguido, configuramos el entorno con la herramienta Diseñador de Esquemas. Los archivos correspondientes al DataSet se crearán en la carpeta App_Code del proyecto, si estamos desarrollando una aplicación Web. A su vez, se iniciará el asistente para conexión con fuente de datos, herramienta que nos permite crear el esquema a partir de una tabla existente en un motor. En este caso, presionaremos la opción Cancelar para cerrarlo.
Para agregar un elemento en el diseñador

> Los esquemas XML permiten realizar DataSets fuertemente tipados a través del diseñador visual.

DataSet, basta con arrastrarlo desde la barra de herramientas. A continuación, agregamos un objeto **DataTable** y establecemos su nombre en **dtClientes**. Esto generará un objeto vacío, sin columnas establecidas.

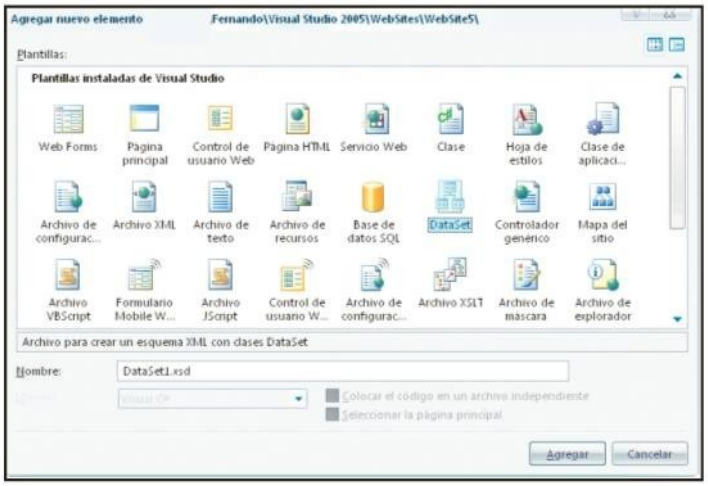


FIGURA 002 | Agregando un DataSet a un proyecto, en Visual Web Developer Express 2005.

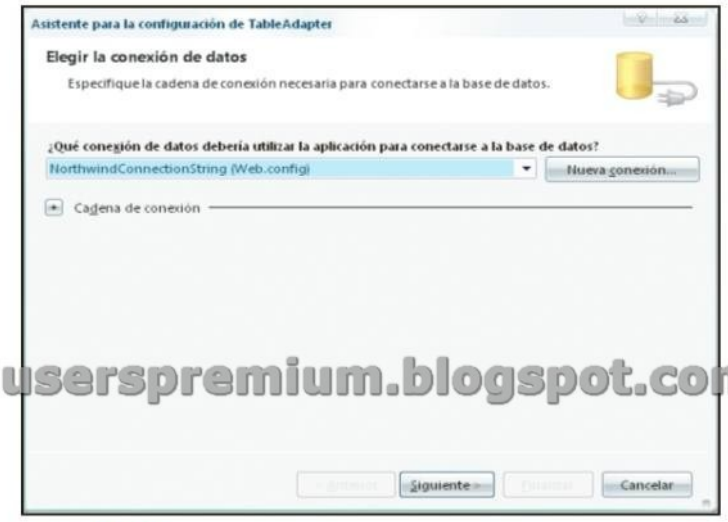


FIGURA 003 | Este asistente permite realizar el esquema en unos sencillos pasos. Más adelante veremos en detalle su funcionamiento.

Para diseñar un DataSet de forma visual agregaremos un nuevo elemento de tipo DataSet a nuestro proyecto.

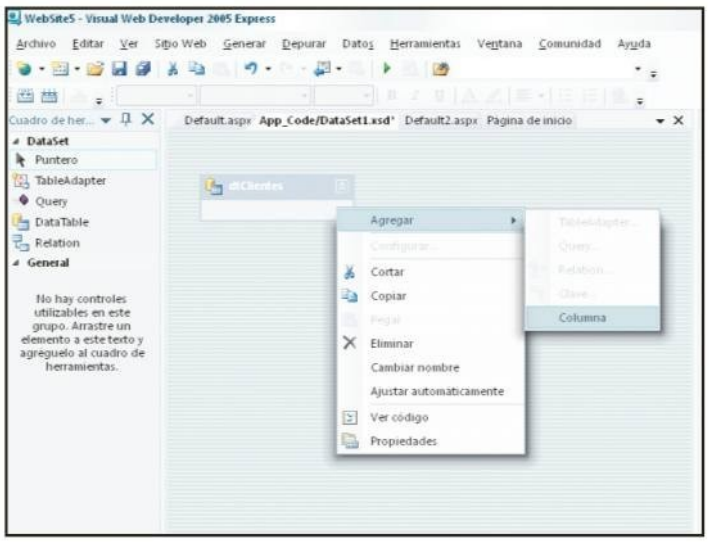


FIGURA 004 | Desde el menú contextual podemos agregar columnas a nuestra tabla de datos, y también establecer claves y relaciones a partir de éstas.

Insertar columnas

Una vez generada la tabla, agregamos las columnas o, mejor dicho, objetos de tipo DataColumn. Para esta tarea utilizamos el menú contextual del objeto dtClientes, y seleccionamos la opción Agregar/Columna. Al agregarla, podremos establecer su propiedad Name, su tipo de dato en formato .NET DataType, la posibilidad de aceptar valores nulos de la base de datos (AllowDBNull), qué acción realizar en caso de que se trate de insertar un registro con este campo en valor nulo (NullValue) y un valor por defecto, DefaultValue, entre otros aspectos.

Cadena de conexión

Una cadena de conexión es el elemento principal para establecer un vínculo con el motor de base de datos. En ella se incluyen los datos necesarios (usuario, contraseña, límite de tiempo, opciones de seguridad, etc.), concatenado por punto y coma ";" en pares de Clave-Valor:

```
Data Source=CadConex;Initial Catalog=
northWind;User Id=fluna; Password=1234;
```

Tabla 1 | Componentes de ConnectionString

Clave	Valor / Descripción
DataSource	Instancia de SQL Server a la cual queremos conectarnos. Puede estar formada por el nombre de la PC en donde se encuentra el servidor, seguido del nombre de la instancia (\\Informatica\NorthWind) o sólo el nombre de la PC (Informatica), si éste no tiene otras instancias de SQL Server instaladas.
DataBase	Nombre de la base de datos a la que deseamos conectarnos. Debemos tener permiso de acceso a la base seleccionada en el motor SQL Server.
Connection Timeout	Es el tiempo máximo en segundos para esperar la devolución de un pedido al motor.
User ID	Nombre de usuario con el cual queremos conectarnos al motor SQL Server.
Password	Contraseña de dicho usuario.
Integrated Security	Permite especificar si utilizaremos o no una conexión integrada de Windows. Si su valor es true, se utilizará el usuario logueado o el establecido en Internet Information Server (en caso de una aplicación Web).

UNA CADENA DE CONEXIÓN ES EL ELEMENTO PRINCIPAL PARA ESTABLECER UN VÍNCULO CON EL MOTOR DE BASE DE DATOS. EN ELLA SE INCLUYEN LOS DATOS NECESARIOS CONCATENADOS POR PUNTO Y COMA ";" EN PARES DE CLAVE-VALOR.

Objetos Connection

La cadena de conexión es necesaria para instanciar objetos del tipo Connection (ya sea SQLConnection, OracleConnection, OleDbConnection, etc.) y sus propiedades serán contenedoras de los valores que indicamos en dicha cadena de conexión.

Conexión OleDB

Las cadenas OleDB poseen pequeñas diferencias con sus pares de SQL Server, y su composición dependerá, exclusivamente, del proveedor de acceso que utilicemos.

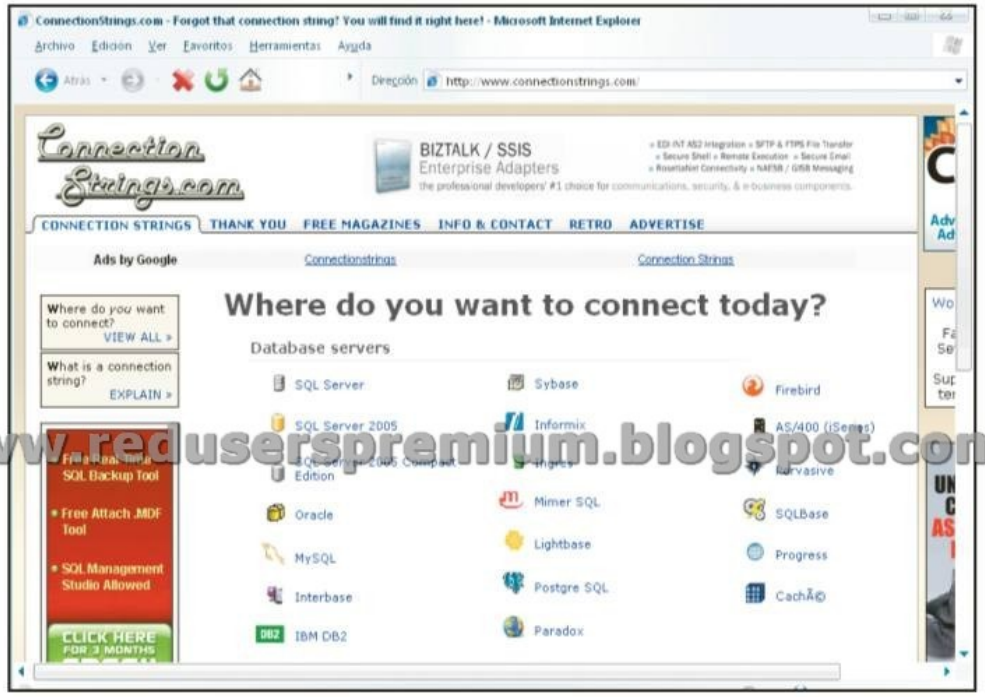
Conexión a SQLServer

Como nuestro objetivo en el curso es conectarnos a un motor de datos SQL Server 2005, debemos incluir claves y valores especificados en la Tabla 1.

¿Nos olvidamos de la cadena? No hay problema

Para no tener que recordar de memoria qué claves y valores componen la cadena de conexión de cada motor/versión de base de datos, podemos visitar el sitio www.connectionstrings.com. Está en inglés, pero es intuitivo, y presenta en forma de iconos cada uno de los motores disponibles.

FIGURA 005 | La única conexión que necesitamos recordar está en www.connectionstrings.com.



System.Data

A continuación veremos las clases que componen el espacio de nombres System.Data.

DataSet

La clase DataSet permite contener una colección de tablas en memoria. Puede relacionarse con un esquema XML, con la extensión XSD para cargar todos los objetos que lo componen (tablas, relaciones, claves, etc.). Tiene uno o más objetos DataTable accesibles desde la propiedad Tables.

DataTable

Representa una colección concreta de filas y columnas, que contiene los objetos DataRow y DataColumn, respectivamente. A su vez, posee lógica de serialización embebida (guardado en XML).

DataRow

Equivale a una fila de datos en memoria. Puede haber cero o más DataRow en un objeto DataTable. Contiene los datos concretos de la fila.

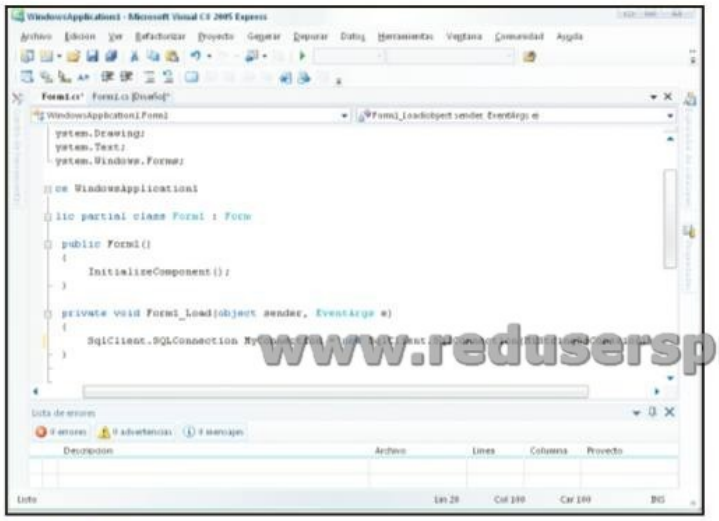


FIGURA 006 | Aquí vemos el entorno de desarrollo C# utilizando el objeto SqlConnection.

DataColumn

Representa una columna, dentro de una tabla en memoria. Incluye el tipo de dato, la posibilidad de aceptar o no valores nulos, autoincremento y demás. No contiene datos/valores.

DataType

Enumera los tipos de datos disponibles para columnas en memoria.

DataReader

Permite acceso de sólo lectura a datos, sin buffer ni posibilidad de mantenerlos en memoria.

Constraints

Estos objetos permiten establecer restricciones entre uno o más objetos DataColumn.

DataRelation

Este objeto representa una relación padre/hijo entre objetos DataTable, especificando columnas primaria y secundaria para establecer la misma.

DataView

Un objeto DataView equivale a una Vista en el modelo transaccional, y permite tener una lectura personalizada de datos almacenados en un objeto DataTable, sin modificarlo. Es posible aplicar filtros, órdenes y agredado de filas, como si se tratase del objeto DataTable original.

DataException

Esta clase es una heredada de Exception, y se utiliza para informar errores ocurridos en la ejecución de métodos de acceso a datos.

System.Data.SQLClient

Aquí encontraremos las clases necesarias para conectarnos con motores SQL Server, desde la versión 2000 en adelante. Las implementaciones concretas, como SqlDataAdapter, SqlConnection o SqlParameter, cumplen la misma funcionalidad explicada en System.Data.Common. Más adelante utilizaremos este espacio de nombres para ejemplificar el proceso de consulta y actualización de datos.

Creando una conexión

El objeto SqlConnection nos permite establecer un vínculo con el motor de base de datos, a partir de la cadena de conexión (o Connection String). Para poder instanciarlo, usaremos las siguientes líneas:

```
'Codigo Visual Basic
Dim MyConnection as New SqlClient.
SqlConnection(MiStringDeConexion)

//Codigo C#
SqlClient.SQLConnection MyConnection = new
SqlClient.SqlConnection(MiStringDeConexion)
```

Creación de comandos

Un comando SqlCommand nos permite ejecutar una instrucción T-SQL (Transact SQL) contra el motor de base de datos que fue establecido en la conexión, y devolver el resultado en objetos. Para crearlo, es necesario pasar, mediante un parámetro, la conexión SqlConnection y una variable del tipo String, con el comando por ejecutar, según como vemos a continuación:

```
'Codigo Visual Basic
Dim consulta as string = "SELECT * FROM
MiTabla"
Dim MyCommand as new SqlClient.SqlCommand
(consulta,MyConnection)

//Codigo C#
String consulta = "SELECT * FROM MiTabla"
SqlClient.SqlCommand MyCommand = new
SqlClient.SqlCommand(consulta,MyConnection)
```

> Los objetos dentro de System.Data permiten representar un modelo relacional en objetos.

Ejecutar comandos, obtener resultados

Una vez que fue configurada la consulta en el objeto SqlCommand y la correspondiente instancia del objeto SqlConnection, llega el momento de ejecutar el comando y guardar los resultados en un nuevo objeto para su navegación. Si sólo precisamos acceder a los datos para su lectura (recorrerlos hacia adelante una única vez), nos convendrá utilizar el objeto DataReader para obtener este resultado. En cambio, si queremos mantener toda la estructura de datos en memoria y retornarla a capas superiores, será necesario el objeto DataSet.

SqlDataReader

Este objeto posee algunas peculiaridades que debemos conocer a la hora de utilizarlo.
Para empezar, funciona como un lector secuencial de datos, y se desplaza de a un registro por vez. Posee un método llamado Read, encargado de posicionar nuestro SqlDataReader con el primer registro y, así, avanzar de a uno. Esto retorna un booleano positivo mientras sea posible avanzar en la lectura. Por este

System.Data.OracleClient contiene los objetos para acceso a bases de datos Oracle, con clases similares a las aquí expuestas.

motivo, toda nuestra lógica será escrita dentro de un bucle While, teniendo como condición lógica el resultado de este método:

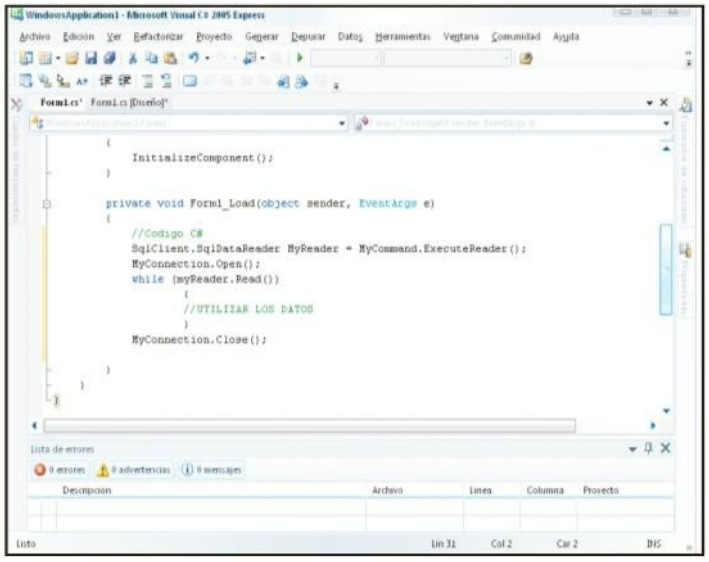


FIGURA 007 | El entorno de desarrollo de Visual C# utilizando el objeto SqlDataReader.

System.Data.OleDb

El acceso a datos genérico a través de OleDb se realiza con los objetos incluidos en este espacio de nombres. Más adelante, veremos aquellos objetos específicos para conexiones del tipo OleDb y su correspondiente cadena de conexión (las diferencias y semejanzas con su par SQL).

```
'Codigo Visual Basic
Dim MyReader as SqlClient.SqlDataReader()
MyReader = MyCommand.ExecuteReader()
'Abrimos la conexión SQL
MyConnection.Open()
While MyReader.Read()
  'UTILIZAR LOS DATOS
End While
MyConnection.Close()

//Codigo C#
SqlClient.SqlDataReader MyReader = MyCommand.
ExecuteReader();
MyConnection.Open();
while (myReader.Read())
    {
      'UTILIZAR LOS DATOS
    }
MyConnection.Close();
```

Lógica en el servidor

Si escribimos nuestra lógica en forma de procedimientos almacenados (Stored Procedures) dentro del motor de base de datos, y éstos requieren de parámetros específicos para su funcionamiento, tendremos que utilizar objetos particulares para invocar dichas funciones y establecer los valores de los parámetros mencionados.

Llamar a un procedimiento almacenado

Para llamar a un procedimiento almacenado debemos configurar nuestro objeto SqlCommand con el nombre del procedimiento en cuestión e indicar que el tipo comando corresponde al valor StoredProcedure de la enumeración CommandType, como vemos en el código a continuación:


```
'Codigo Visual Basic
Dim MySPCommand as new SqlCommand
MySPCommand.CommandType = CommandType.
StoredProcedure
```

```
'Codigo C#
SqlCommand MySPCommand = new SqlCommand();
MySPCommand.CommandType = CommandType.
StoredProcedure;
```

Creando parámetros

Es común que los procedimientos requieran distintos parámetros para su funcionamiento, tanto de entrada como de salida. Estos parámetros, representados por el objeto SqlParameter, son el nexo entre el motor SQL Server y el framework .NET.

El objeto SqlParameter posee un constructor con argumento variable. Podremos utilizar la sobrecarga más conveniente para nuestro parámetro, como vemos en el código siguiente:

```
'Codigo Visual Basic
'En esta sobrecarga indicamos solo el nombre
del Parámetro y su valor (recibe cualquiera
de tipo Object o heredado)

Dim Valor As String = "Valor"
Dim myParameter As New SqlClient.
SqlParameter("Nombre", Valor)
```

```
'En esta sobrecarga indicamos nombre, tipo y
longitud
Dim myParameter As New SqlClient.
SqlParameter("Nombre", SqlDbType.VarChar, 20)

'Aquí establecemos nombre y tipo de dato,
pero no el valor
Dim myParameter As New SqlClient.
SqlParameter("Nombre", SqlDbType.VarChar)
```

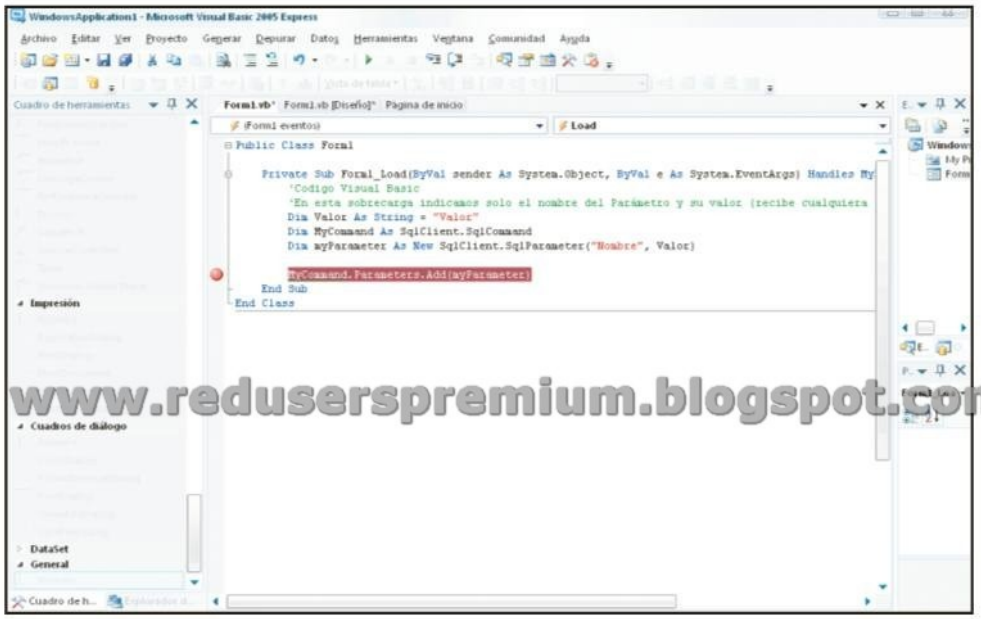
En el caso de las sobrecargas que no establezcan valor para el parámetro, será necesario hacerlo posteriormente desde la propiedad Value:

```
'Codigo Visual Basic
myParameter.Value = "Valor1"
```

Para enviar parámetros al momento de ejecutar comandos, sólo es necesario agregar cada uno de ellos en la propiedad Parameters del objeto de tipo SqlCommand, creado en el código anterior:

```
'Codigo Visual Basic
MyCommand.Parameters.Add(MyParameter)
```

FIGURA 008 | El entorno Visual Basic .NET representando el envío de parámetros.



OleDb

Conoceremos una tecnología vital para el acceso a datos, de manera independiente de la plataforma de base de datos elegida.

Esta tecnología propuesta por Microsoft permite acceder a prácticamente cualquier fuente de datos con el driver correspondiente. El framework .NET nos ofrece un conjunto de clases específicas para su uso.

Objetos System.Data.OleDb

Las implementaciones concretas de los objetos de acceso a datos son equivalentes a las propuestas para SQL Server, dentro del espacio System.Data.SqlClient, exceptuando algunas diferencias concretas.

OleDbConnection

Las conexiones OleDb se manejan a través de objetos OleDbConnection, equivalentes a SqlConnection. Este objeto inicializa las propiedades referentes al proveedor de acceso a datos al momento de instanciarlo. La cadena de conexión de OleDb es particular para cada motor, como vemos en el siguiente ejemplo:

```
'Codigo Visual Basic.Net
'Utilizamos la cadena de conexión para SQL
Server
Dim MyConn as New Data.OleDbConnection
("Provider=SQLOLEDB;Data Source=localhost;
Initial Catalog=MiBaseDePruebas;Integrated
Security=SSPI;")
MyConn.Open()

// El mismo Codigo para el lenguaje C#
//Utilizamos la cadena de conexión para SQL
Server
Data.OleDbConnection MyConn = new Data.
OleDbConnection("Provider=SQLOLEDB;Data
Source=localhost;Initial Catalog=MiBaseDe
Pruebas;Integrated Security=SSPI;");
MyConn.Open();
```

Cadena de conexión OleDb

La cadena de conexión, en el caso de OleDb, podrá variar para cada motor de base de datos, tal como muestra el siguiente ejemplo:

```
'Codigo Vb.Net
'Cadena de conexión para (Oracle Driver de
Microsoft)
Dim myOracleString as String = "Provider
=msdaora;Data Source=MyBaseOracle;User
Id=USUARIO;Password=CONTRASEÑA;"
```

§ OleDbPermission

Esta clase nos permite establecer niveles de seguridad a través de una configuración, sobre conexiones OleDb. Su uso, si bien no es fundamental para el consumo de datos por OleDb, es frecuente en aplicaciones Web, ASP.NET.

PARA ESTABLECER UNA CONEXIÓN CON MOTORES ORACLE A TRAVÉS DE OLEDB, EXISTEN DRIVERS PUBLICADOS POR MICROSOFT (MICROSOFT PROVIDER FOR ORACLE) Y POR ORACLE (ORACLE PROVIDER FOR OLEDB) INDEPENDIENTES.

```
'Cadena para Microsoft Excel
Dim MyExcelString as String = "Provider=
Microsoft.Jet.OLEDB.4.0;Data Source=
C:\MiArchivoExcel.xls;Extended Properties=
"Excel 8.0;HDR=Yes;IMEX=1";"

'Cadena para Microsoft Access
Dim MyAccessString as String = "Provider=
Microsoft.Jet.OLEDB.4.0;Data Source=
C:\MiBaseDeAccess.mdb;User Id=admin;Password=;"

'Cadena para MySql
Dim MyMySqlString as String = "Provider=
MySQLProv;Data Source=mydb;User Id=Usuario
MySql;Password=Contraseña;"
```

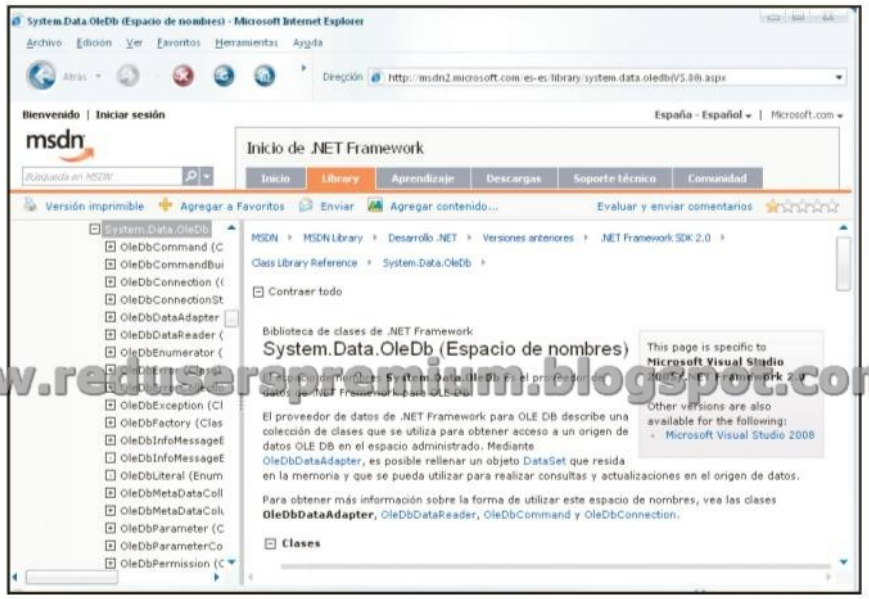
Drivers

Es necesario recordar que, para el correcto funcionamiento de cada conexión mencionada, es preciso tener instalado el driver correspondiente de la base/motor/formato para OleDb.

OleDbParameter

La clase OleDbParameter nos permite el paso de parámetros de entrada/salida hacia cualquier fuente de datos con soporte para ellos. Sus propiedades son equivalentes al objeto SqlParameter visto recientemente.

FIGURA 009 | Toda la ayuda detallada del namespace System.data se encuentra en MSDN.



ALGUNOS PROVEEDORES DE DATOS NO INCLUYEN EN SUS RESULTADOS LA PROPIEDAD DE CLAVE PRIMARIA, POR LO QUE SERÁ NECESARIO CONFIGURARLA EN FORMA MANUAL.

OleDbException

Las excepciones en OleDb pueden ser de lo más variadas, debido a la amplia cantidad de motores/orígenes de datos accesibles. Por este motivo, tenemos que prestar especial atención a ellas y evaluar correctamente el o los objetos de tipo OleDbError que contienen. Si se producen errores en la obtención, modificación o inserción de datos, el objeto OleDbError es el encargado de informarnos, dependiendo del proveedor, de las circunstancias de dicho error. Un objeto OleDbDataAdapter se ocupa de crear y devolver un OleDbError, en caso de que se produzca alguna excepción de datos. De acuerdo con la severidad del error, la conexión de origen puede ser forzada a cerrarse, para evitar la pérdida de información o problemas más graves aún, como la modificación no deseada de registros.

Obtención de datos

A continuación veremos cómo conectarnos a un motor SQL Server, utilizando OleDb.

OleDbDataAdapter

El objeto DataAdapter nos permite establecer comandos para inserción, obtención y actualización de datos en memoria, hacia la base de datos, y viceversa. Para dichas tareas es necesario que especifiquemos los respectivos comandos en el lenguaje del motor elegido. Para mantener los datos obtenidos del motor, utilizamos la clase DataSet, vista anteriormente, y el método Fill del objeto DataAdapter para su llenado. Dicho método utilizará el comando establecido (SelectCommand), recuperará los datos desde el motor y creará, en caso de que sea necesario, todas las tablas y claves primarias (si así lo establecemos) de dicha consulta.

```
'Codigo Visual Basic
'Utilizaremos como ejemplo el objeto MyConn
Dim MySelectCommand as String = "SELECT *
FROM TablaDePrueba;"
'El constructor del DataAdapter recibe en una
de sus sobrecargas un parámetro para el
Comando de Obtención de datos y otro para la
Conexión SQL
Dim MyAdapter as new OleDbDataAdapter
(MySelectCommand, MyConn)
Dim MyDataSet as DataSet
MyAdapter.Fill(MyDataSet)
//Codigo C#
//Utilizaremos como ejemplo el objeto MyConn
string MySelectCommand = "SELECT * FROM
TablaDePrueba";
OleDbDataAdapter MyAdapter = new
OleDbDataAdapter(MySelectCommand, MyConn);
DataSet MyDataSet;
MyAdapter.Fill(MyDataSet);
```


Una vez que se ha cargado el objeto DataSet, podemos utilizar los datos en cualquier control o proceso de negocio.

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

15

Web Services

Infraestructura - WSDL y UDDI

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

ADO.NET avanzado

Procedimientos almacenados
SqlClient - OleDb

ISBN 978-987-1347-43-8
00015
9 789871 347438