بسمه تعالی

آموزش XML

راهنمای استفاده XML در C#.NET

(ویرایش یک)

محقق و نویسنده:

اوژن استوار

مقدمه

در این کتاب قصد داریم به بررسی زبان نشانه گذاری XML بپردازیم، با مفاهیم و تاریخچه این زبان نشانه گذاری آشنا شده و تکنولوژی های مرتبط با آن را بررسی کنیم. برای انجام فعالیت های گوناگون روی اسناد XML ، به بررسی و معرفی کلاس ها و متدهای موجود در C#.NET خواهیم پرداخت و برای هر مورد مثالی را ارائه می دهیم. تمرکز مطالب بر استفاده از XML به وسیله زبان C# است.

به همراه کتاب برنامه ای ارائه می شود که کلیه ی مثال های بررسی شده در کتاب به زبان C# در آن نوشته شده است. در بعضی از قسمت ها مطالبی برگرفته از وب سایت های فارسی و انگلیسی استفاده شده است، لینک منابع در بخش لینک های نرم افزار قرار دارد، علاقه مندان به این زبان نشانه گذاری جهت یادگیری و تسلط بیشتر بر مفاهیم آن می توانند منابع معرفی شده را مطالعه نمایند.

در قسمت هایی از کتاب که ترجمه متون انگلیسی درج شده است، از ترجمه کلیه واژگان خودداری شده، این امر به دلیل جلوگیری از گمراه شدن ذهن خواننده از اصل موضوع و آسانی در فهم مطالب صورت گرفته است. بعلاوه در پایان هر قسمت توضیح مختصری به زبان انگلیسی درج گردیده است.

کلیه حقوق مادی و معنوی این کتاب و نرم افزار، متعلق به نویسنده آن می باشد. استفاده از مطالب و ارائه نرم افزار فقط با ذکر نام نویسنده و منبع آن (www.ostovarit.com) مجاز است.

با آرزوی موفقیت برای تمامی برنامه نویسان فارسی زبان

اوژن استوار

بخش ها

بخش اول : XML چیست؟ (در این بخش نحوه ساخت سند XML آموزش داده می شود)

بخش دوم : تعاریف و اصطلاحات (در این قسمت با اصطلاحات و تعاریف مرتبط با XML آشنا خواهید شد)

بخش سوم : پیش نویس ها (Schemas) (توضیحاتی برای ساخت و ویرایش فایل های DTD و Xml-Schema)

بخش چهارم : زبان های تعریف سَبک (StyleSheet) (توضیحاتی برای ساخت و ویرایش فایل های CSS و XSL { XSLT ، XPATH و XSL-FO})

بخش پنجم : پردازش اسناد XML (معرفی پارسر های XML {DOM و SAX})

بخش ششم : XML در .NET (معرفی و ارائه مثال از کلاس های : XmlTextReader ، XmlNode ، XmlDocument ، XmlConvert ، XmlWriter ، XmlWriterSettings ، XmlTextWriter ، XslTransform ، XsltSettings ، XmlSchema ، XmlReaderSettings ، XmlValidatingReader ، XmlSchemaCollection ، XmlSchemaSet ، XmlSerializer ، XPathDocument ، XPathNavigator و کلاس های مرتبط دیگر)

بخش هفتم : چند مثال کاربردی (مثال هایی با عنوان : خبر خوان { Rss Reader} ، نمایش و ویرایش یک سند XML در DataGridView ، نمایش یک سند XML در WebBrowser ، تلفیق دو سند XML ، خروجی از پایگاه داده به فرمت XML ، کتابخانه Virtual Token Descriptor)

بخش اول

XML چیست؟

SGML (Standard Generalize Markup Language) سرچشمه XML بوده و در سال ۱۹۸۸ استاندارد شده است. HTML (HyperText Markup Language) زبانی است که در ابتدا با استفاده از SGML تعریف گردید. XML یک زبان نشانه گذاری مشابه HTML نمی باشد. XML زیر مجموعه ای از SGML است (مکانیزمی برای تعریف زبان های نشانه گذاری).

XML (eXtensible Markup Language) یک قالب ذخیره سازی مناسب برای داده ها است که برای مبادله اطلاعات بین برنامه های نامتجانس، بنگاه های تجاری و بانک های اطلاعاتی استفاده می شود. یکی از مزیت های مهم XML قابلیت اجرای آن در پلتفرم های مختلف است. شما می توانید با یک ویرایشگر متن فایل های XML را بررسی کرده و اطلاعات آن را تغییر دهید و به آسانی آن را اشکال زدایی کنید. اما همین امر باعث شده فایل های XML فضای زیادی را اشغال کنند.

از دیگر مزیت های XML درک سریع و آسان آن برای تمامی افراد است. این زبان نشانه گذاری رایگان است. برنامه نویسان از خصوصیات بی نظیر آن برای ایجاد ارتباط و یکپارچگی بین برنامه های تجاری استفاده می کنند. در زیر نمونه ای از یک سند XML را به نمایش می گذاریم:

```
<company>
        <name>Ozhan Ostovar</name>
        <address>Satarkhan</address>
        <city>Tehran</city>
        <country>Iran</country>
</company>
```

بلاک های ایجاد شده در یک سند XML در عین سادگی، دارای انعطاف و قدرت لازم به منظور حمایت از روش های متفاوت مدل سازی اطلاعات می باشند.

XML became a W3C Recommendation February 10, 1998. It was designed to transport and store data. HTML was designed to display data. XML is not a replacement for HTML. XML does not DO anything. It was created to structure, store, and transport information. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it. XML is now as important for the Web as HTML was to the foundation of the Web. It is the most common tool for data transmissions between all sorts of applications.

چرا باید XML را آموخت؟

برای این امر دلایل زیادی وجود دارد. بهترین راه برای درک اهمیت آموختن XML نگاهی به توانایی های آن است در اینجا اشاره ای به برخی از استفاده های آن خواهیم داشت.

استفاده از پایگاه داده نظیر SQL Server و Access و Oracle و ... در بین برنامه نویسان رایج است. اما استفاده از آنها برای ذخیره اطلاعات همواره مناسب نخواهد بود. افزایش حجم و زمان بری نصب از معایب استفاده از پایگاه داده بزرگ در برنامه های کوچک محسوب می شود، در صورتی که XML با سرعتی بالا و بدون نیاز به نصبِ پیش نیاز ها (requirements) مربوط به پایگاه داده این مشکل را بر طرف می سازد.

ایجاد ارتباط و یک پارچگی بین نرم افزار ها از فواید بزرگ XML به حساب می آید. یک نرم افزار حسابداری که کلیه مشتریان شما در آن تعریف شده است را در نظر بگیرید در طرف دیگر نرم افزاری برای ارسال پست الکترونیک گروهی دارید که هیچ اطلاعاتی از پست الکترونیک مشتریان در آن ثبت نشده. بسیار آسان خواهد بود اگر با چند کلیک از نرم افزار اول خروجی با فرمت xml تهیه شده و در آن نام مشتری همراه با پست الکترونیک آن ذخیره گردد سپس در نرم افزار دوم با استفاده از فایل مذکور بدون صرف زمان و هزینه اضافی اطلاعات مشتریان ذخیره و مورد استفاده قرار گیرد.

با آنکه وب یکی از سریعترین رشد های مشاهده شده در تاریخ بشر را تجربه میکند اما هنوز طفلی نو پا بیش نیست. هر ابزاری که بتواند در این دنیای مجازی از پس کاری برآید به سرعت نظرها را به خود جلب خواهد کرد. یکی از گرایش های مهم موجود در اینترنت تجارت الکترونیک است. رشد سریع تجارت الکترونیک اکنون با موانعی روبرو شده است. چون کار آنقدرها هم که در ابتدا تصور می شد ساده نبود. با این همه، تردید نباید داشت که تجارت الکترونیک بخش مهمی از اقتصاد کنونی جهان است. نکته کلیدی در تجارت الکترونیکی، جمع آوری و ذخیره سازی و پردازش و تفسیر اطلاعات پراکنده وب است. و این کاریست که به خوبی از عهده XML بر می آید.

از دلایل دیگری که XML برای طراحان وب محبوب شده، جدا کردن محتوا از ظاهر می باشد. به این معنا که شما اطلاعات را در یک فایل XML نگهداری می کنید و در زمان نمایش در یک قالب تهیه شده (مثلا HTML) آن را به نمایش می گذارید. پس اگر بخواهید اطلاعات را به روز نمایید کافی است به فایل XML مورد نظر رجوع کنید و نگران نحوه نمایش آن نباشید.

ساخت یک سند XML

یادگیری ایجاد یک سند XML بسیار آسان می باشد. به طوری که با مشاهده یک سند قادر به تشخیص قواعد ایجاد آن خواهید بود. در اینجا قصد دارم تا یک سند XML را تشریح کنم. در بخش بعد به طور کلی اصطلاحات استفاده شده توضیح داده می شود. در زیر یک سند XML داریم:

```
<?xml version="1.0" encoding="utf-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

در خط اول version سند نوشته می شود و در ادامه encoding سند را مشخص می کنیم در صورتی که از utf-8 استفاده شود کلیه کاراکتر های فارسی پشتیبانی می شود. خط اول معرف فایل می باشد و تگ پایانی ندارد. در خط بعدی، ریشه یا همان Root قرار دارد که نشان دهنده محتویات سند است مثلا در مثال ما <note> ریشه سند می باشد. و در چهار خط بعدی چهار فرزند برای آن تعریف شده است که <to>، <from>، <heading>، <body> نام دارند. نکته ای که درباره ی هر element باید در نظر داشته باشید این است که بر خلاف html حتما باید تمامی تگ ها ایجاد شده دارای تگ پایانی باشند.

هر یک از element ها می تواند شامل اطلاعاتی درباره خود باشند که در attribute ها نوشته می شوند. این اطلاعات metadata به معنای اطلاعاتی در باره ی اطلاعات خوانده می شوند مانند:

```
<note date="12/11/2002">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

همان طور که ملاحظه میکنید در خط اول مقدار تاریخ به عنوان یک attribute ذخیره شده است. حال همین ساختار را به شکل زیر پیاده سازی میکنیم:

```
<note>
  <date>12/11/2002</date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

هر دو مثال حاوی اطلاعاتی مشابه هستند اما در مثال اول تاریخ به عنوان یک attribute ذخیره شده بود و در مثال دوم خود یک element مستقل می باشد. برای اینکه یک داده به چه صورت تعریف شود قانون مشخصی وجود ندارد اما توصیه می شود که داده ها به صورت element های مستقل تعریف شوند زیرا در زمان تعریف و فراخوانی محدودیتی وجود نخواهد داشت. و اطلاعاتی که در attribute ها می آیند بهتر است از نوع Metadata ها باشند، نه اطلاعات اصلی. برای مثال:

```
<User>
  <NAME xmlns="FirstName and LastName">Ozhan Ostovar</NAME>
  <E-mail xmlns="Just yahoo mail"> ostovarit@yahoo.com</E-mail>
```

```
  <Mob xmlns="+98">0912</Mob>
  <Image xmlns="jpg">myPic.jpg</Image>
</User>
```

حال مثال اول خود را به صورت زیر تغییر می دهیم:

```
<note>
  <date>
    <day>12</day>
    <month>11</month>
    <year>2002</year>
  </date>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

حالت فوق بهترین شکل یک فایل XML استاندارد می باشد.

بخش دوم

تعاریف و اصطلاحات

Markup

راهی برای معرفی شکل و شمایل یک متن است مانند HTML که یک زبان Markup است و روش نمایش متون را در مرورگرها مشخص می کند. برخلاف تصور عموم، Xml یک نوع Markup نیست بلکه ابزاری برای تعریف Markup است. گاها به Markup، زبان یا Language هم گفته می‌شود. قدمت استفاده از Markup به قبل از کامپیوتر بر می گردد. مثلا در دنیای نشر از علائم خاصی در متن های ویرایش شده استفاده می شد تا به پردازنده متن، (انسان یا ماشین) اعلام شود چه نوع عملیاتی را در رابطه با اطلاعات می بایست انجام دهد.

Tag

به متنی که بین دو علامت کوچکتر و بزرگتر قرار دارد (به همراه خود علائم) تگ گفته می‌شود. مثلا <name> و یا </country>. بر خلاف HTML که در صورت فراموش شدن درج تگ پایانی مرورگر بدون مشکل و بروز خطا فایل را اجرا میکند، در XML همه عناصر باید شامل تگ پایانی باشند.

نکته: تگ های XML به حروف کوچک و بزرگ حساس هستند برای مثال <Note> با <note> برابری نمی کند.

Content

به هر گونه متنی که بین دو tag قرار دارد گفته می‌شود. <FirstName>**Name**</FirstName>

Element

مجموعه کامل tag و content آن را element می‌گویند. <city>**Tehran**</city>

نکته: هر element می‌تواند شامل element های دیگری باشد. تو در تو بودن element ها می‌تواند تا هر اندازه ای که لازم باشد تکرار شود element ها نمی‌توانند ه‌م پوشانی داشته باشند. یعنی تگ آغاز و پایان هر element باید به طور کامل بین تگ آغاز و پایان element والد خود باشد.

Attribute

یعنی اطلاعاتی که داخل tag به منظور توضیح رفتار آن element قرار می‌گیرد. این اطلاعات معمولا اطلاعاتی درباره اطلاعات یا همان Metadata ها هستند.

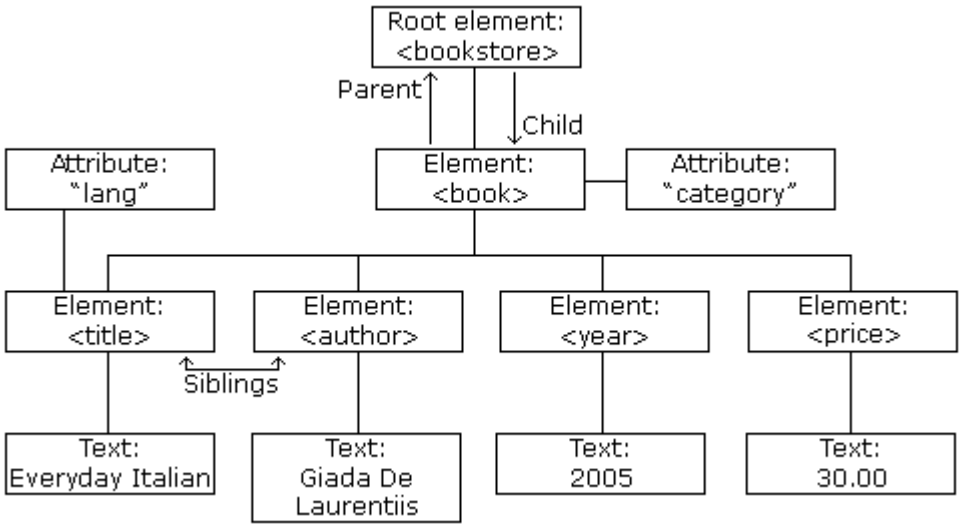
<E-mail xmlns="**Just yahoo mail**">ostovarit@yahoo.com</E-mail>

Node

به هر کدام از مفاهیم tag ، element و attribute یک Node گفته می‌شود.

Root Element

به element ی که دربرگیرنده همه element های یک سند xml است گفته می‌شود. هر سند xml فقط یک Root element دارد.



درخت بالا نمایش سند XML زیر است :

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

CDATA

راهی است برای وارد کردن Content های طولانی که شامل کاراکترهای غیر مجاز و خاصی هستند. مثلا شما هیچ وقت نمی‌توانید در content یک element علامت «کوچکتر از» یعنی "<" را وارد کنید بلکه به جای آن باید از < استفاده شود. برای درج متنی که علامت‌ها و کاراکترهای خاص دارد از CDATA استفاده می شود. برای مثال:

```
<?xml version="1.0" encoding="utf-8" ?>
<script>
<![CDATA[
function matchwo(a,b)
{
if (a < b && a < 0) then
  {
  return 1;
  }
else
  {
  return 0;
  }
}
]]>
</script>
```

The term CDATA is used about text data that should not be parsed by the XML parser. Characters like "<" and "&" are illegal in XML elements. Some text, like JavaScript code, contains a lot of "<" or "&" characters. To avoid errors script code can be defined as CDATA.

لیست کاراکتر های ویژه و معادل آنها را در زیر مشاهده آماده است:

less than	<	<
greater than	>	>
ampersand	&	&
apostrophe	‘	'
quotation mark	“	"

White Space

به مجموعه کاراکترهای CR(0x0D) و Tab(0x09) و LF(0x0A) گفته می‌شود.

نکته : Xml Processor ها در شرایط مختلف، برخوردهای متفاوتی با white space ها دارند. مثلا بین هر دو attribute هر تعداد هم white space وجود داشته باشد مشکلی در ساختار xml به وجود نمی‌آید. این یعنی هر attribute را می‌توان در یک خط جداگانه تعریف کرد. در نقطه مقابل، white space های موجود در content یک element حفظ شده و در نظر گرفته می‌شوند. این یعنی اگر content یک element از چندین خط نوشته تشکیل شده بود در content مربوطه کلیه کاراکترهای CR/LF هم ذخیره شده و در نمایش‌های بعدی (مثلا درج اطلاعات xml در یک دیتابیس) هم نمایش داده خواهند شد.

Metadata

به معنای اطلاعاتی درباره اطلاعات است. برای مثال در سند زیر مقدار ایدی برای هر element یک متادیتا می باشد

```
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

Namespace

هر سند xml شامل یک سری tag و attribute است به علاوه یک ساختار کلی که نشان می‌دهد اجزای آن چطوری با هم ارتباط دارند. این اجزا را Vocabulary و ساختار آنها را Grammar می‌نامند. به مجموعه‌ای از vocabulary های مرتبط با هم که یک زبان یا markup مشخص را تعریف می‌کنند namespace گفته می‌شود. namespace راه حلی برای مشکل تصادم اسامی است.

namespace ها به صورت یک attribute تعریف می‌شوند ns (مختصر شده namespace) را هم می‌توان با نام و هم بدون نام تعریف کرد. مثال:

```
<Letter xmlns:Tehran="www.yourcompany.com/test1">
```

In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications. An XML parser will not know how to handle these differences. XML Namespaces provide a method to avoid element name conflicts.

بخش سوم

پیش نویس ها (Schemas)

از پیش نویس ها برای بررسی صحت محتویان یک سند، ایجاد مقادیر پیش فرض برای element ها، کمک در خواندن و ویرایش یک سند XML و ترجمه از فرمت فعلی XML به فرمتی دیگر استفاده می گردد.

در این راستا می توان کد هایی نوشت که کلیه عملیات فوق را انجام می دهند، اما کد های نوشته شده محدود به یک سند خاص خواهند بود. اطلاعات موجود در پیش نویس این امکان را به وجود می آورند که با نوشتن یک پردازنده پیش نویس، از آن در سند های متفاوت بدون نیاز به برنامه نویسی مجدد استفاده کرد.

XML برای تعریف پیش نویس ها (Schemas) از دو زبان استفاده می کند: DTD و XML-Schema

DTD

راهی برای تشخیص اینکه یک سند xml مطابق با یک زبان یا markup می باشد یا خیر. این اعتبار سنجی از منظر واژگان (Vocabulary) و ساختار (Grammar) بررسی می‌شود. DTD رقیبی به نام Xml Schema دارد. DTD از Schema قدیمی تر است و برای اعتبارسنجی اسناد SGML استفاده می شده در حالی که XmlSchema جدیدتر، راحت‌تر و رایج‌تر است.

برای مثال فایل XML زیر:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

به فایل DTD اشاره میکند که محتوای آن به شرح زیر است :

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

همان طوری که در مثال فوق می بینید یک سند DTD مکانیزمی برای تعریف پیش نویس ها می باشد که محوریت آن بر سند استوار است. گرامر استفاده شده در آن اختصاصی بوده و ابزارهای اندکی به منظور پردازش آنها وجود دارد.

A Document Type Definition (DTD) defines the legal building blocks of an XML document. It defines the document structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML document, or as an external reference.

در ادامه قصد داریم یک فایل DTD ایجاد نماییم. جهت ایجاد فایل می توانید از یک ویرایشگر متن ساده استفاده نمایید و کد های DTD را داخل آن بنویسید. در اسناد XML ساده ترین راه ایجاد DTD داخل خود سند می باشد. برای این منظور بالای سند پس از اعلان XML عبارت <!DOCUMENT root[….]> را تایپ می کنیم. و محتویات DTD را به جای نقطه چین جایگذاری می کنیم. element هایی که ایجاد می شوند می توانند شامل یک متن، element های دیگر و یا خالی باشند.

برای ایجاد هر element در یک سند DTD تایپ کنید <!ELEMENT *TagName* که در اینجا TagName نام تگ مورد نظر می باشد. اگر عنصر مورد نظر محتویات ندارد در ادامه واژه ی Empty را تایپ کنید در غیر این صورت واژه (content) را تایپ نمایید که منظور توضیحاتی درباره عنصر یا متن می باشد. در صورتی که element مورد نظر ترکیبی از عناصر و متن باشد کلمه ANY را وارد نمایید.

```
<!ELEMENT TagName EMPTY>
```

نکته: DTD ها مشخص می کنند که هر element می تواند چه مقداری داشته باشید اگر برای تمامی element ها از ANY استفاده کنید آنگاه سند DTD شما مفهومی نخواهد داشت و نیازی به ایجاد آن نخواهد بود، زیرا که استفاده از اسناد DTD اجباری نیست.

نکته: اگر برای element از واژه ANY استفاده شود آنگاه مقداری آن نمی تواند خارج مقادیری باشد که در DTD تعریف شده است.

مانند:

```
<!ELEMENT TagName ANY>
```

مقدار (#PCDATA) برای یک element مشخص می کند که محتویات آن فقط به صورت متنی بوده و این متن می تواند شامل اعداد، علامت ها و ... باشد در نتیجه element ی که شامل آن باشد نمی تواند از element های دیگر تشکیل شود.

مانند:

```
<!ELEMENT TagName (#PCDATA)>
```

در صورتی که element مورد نظر دارای یک زیر مجموعه باشد. برای معرفی آن به صورت زیر عمل می کنیم و بجای کلمه ChildName مقدار Child Element را قرار می دهیم.

```
<!ELEMENT TagName (ChildName)>
```

برای ترتیب دهی به زیر مجموعه های یک element به صورت زیر عمل میکنیم:

```
<!ELEMENT TagName(ChildName1, ChildName2 , ...)>
```

کاراکتر | به معنای "یا" عمل می کند یعنی در صورتی که برای تعریف element ی از آن استفاده شود آن element می تواند یکی از نوع های طرفین کارکتر | را قبول نماید. به مثال زیر توجه نمایید:

```
<!ELEMENT TagName(ChildName1 | ChildName2)>
```

در مثال فوق مقدار زیر مجموعه TagName یکی از مقادیر ChildName1 یا ChildName2 می تواند باشد.

برای مشخص کردن تعداد دفعات تعریف هر Child در یک element در زمان تعریف آنها می توان از علامت های + به معنای تعریف حداقل یک بار (اجباری) و * به معنای تعریف هیچ یا بی نهایت (کاملا اختیاری) و ؟ به معنای تعریف حداکثر یک بار استفاده کرد.

```
<!ELEMENT TagName(ChildName1+, ChildName2* , ...)>
```

در اسناد XML ، Attribute هر element ایجاد نمی شوند مگر اینکه در DTD مربوطه تعریف شده باشند. برای مثلا برای فایل XML زیر:

```
<book cat="History">
  <myBook>IranHistoey</myBook>
</book>
```

در DTD این چنین تعریف میکنیم:

```
<!ELEMENT book (#PCDATA)>

<!ATTLIST book cat CDATA #IMPLIED >
```

همان طور که میبینید ابتدا کلمه <!ATTLIST و بعد element اصلی و پس از آن Child آمده است سپس نوع داده مشخص شده و کلمه #IMPLIED که به معنای دلبخواهی بودن Attribute می باشد نوشته می شود. اگر مقدار Attribute شامل مقادیر گوناگون باشد عبارت #REQUERMENT نوشته خواهد شد. و اگر مقدار آن ثابت باشد از #FIXED value استفاده میکنیم.

برای مشخص کردن Attribute هایی که مقدار آنها برابر با ایدی و یا هر مقدار منحصر به فردی باشد از کلمه IDREF استفاده می شود. اگر مجموع چند ایدی را در بر داشته باشد از IDREFS استفاده میکنیم. مانند:

```
<!ATTLIST book cat IDREF #REQUERMENT >
```

یک نمونه فایل DTD با توجه به اطلاعات ارائه شده در این بخش:

```
<!DOCTYPE TVSCHEDULE [

  <!ELEMENT TVSCHEDULE (CHANNEL+)>
  <!ELEMENT CHANNEL (BANNER,DAY+)>
  <!ELEMENT BANNER (#PCDATA)>
  <!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
  <!ELEMENT HOLIDAY (#PCDATA)>
  <!ELEMENT DATE (#PCDATA)>
  <!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
  <!ELEMENT TIME (#PCDATA)>
  <!ELEMENT TITLE (#PCDATA)>
  <!ELEMENT DESCRIPTION (#PCDATA)>

  <!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
  <!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
  <!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
  <!ATTLIST TITLE RATING CDATA #IMPLIED>
  <!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

با اطلاعاتی که در این قسمت مطالعه کردید توانایی لازم جهت ایجاد، ویرایش و درک فایل های DTD را بدست آوردید. در فصل های ششم روش استفاده از فایل های DTD را در یک نرم افزار به کمک C# خواهیم آموخت. در صورت تمایل برای کسب اطلاعات بیشتر در مورد DTD به لینک های معرفی شده در بخش لینکهای نرم افزار مراجعه فرمایید.

(XSD) Xml-Schema

راهی است برای اعتبار سنجی یک سند XML. این اعتبار سنجی از دو دیدگاه واژگان (Vocabulary) و ساختار (Grammar) انجام می‌شود. Xml Schema رقیبی جدی و در خیلی از موارد جایگزینی برای DTD محسوب می‌شود. XML-Schema یک استاندارد جدید بوده که مختص XML طراحی شده است. این تکنولوژی محدودیت های DTD را برطرف نموده و اکثر ابزار های تولید شده توسط عرضه کنندگان معتبر، مجهز به این تکنولوژی می باشد. به افرادی که تازه به دنیای XML وارد شده اند، توصیه می شود که گرامر DTD را فراموش و خود را با استاندارد XML-Schema وفق دهند.

یک فایل XML-Schema شامل موارد زیر است :

- تعریف element هایی که می تواند در فایل XML باشد.
- تعریف attribute هایی که می تواند در فایل XML باشد.
- مشخص کردن element هایی که می توانند در بر گیرنده element های دیگر باشند.
- مشخص کردن ترتیب و تعداد element های Child.
- مشخص کردن آنکه یک element می تواند خالی یا حاوی متن باشد.
- نوع داده ها در element ها و attribute ها
- مقدار پیش فرض و یا اجباری برای یک element یا attribute

و...

مثالی را که در DTD آورده ایم به خاطر بیاورید، برای نوشتن پیش نویس فایل مجزایی ایجاد کردیم و محدودیتی که در معرفی نوع داده ها داشتیم، حال برای همان فایل XML، پیش نویسی با XML-Schema می نویسیم:

XML :

```
<?xml version="1.0"?>

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

XML-Schema :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.ostovarit.com"
xmlns="http://www.ostovarit.com"
elementFormDefault="qualified">

  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

XML Schema is an XML-based alternative to DTD. An XML schema describes the structure of an XML document. The XML Schema language is also referred to as XML Schema Definition (XSD) The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD. Very soon XML Schemas will be used in most Web applications as a replacement for DTDs. One of the greatest strength of XML Schemas is the support for data types. XML Schemas are written in XML. XML Schema became a W3C Recommendation 02. May 2001.

اکنون طریقه ساخت یک فایل Schema را برای یک سند XML شرح می دهیم:

فایل های XML-Schema با پسوند xsd ذخیره می شوند. در ابتدای سند ریشه اصلی را به شکل زیر تعریف می نماییم:

```
<?xml version="1.0"?>

<XSD:schema>
...
...
</XSD:schema>
```

ریشه اصلی ممکن است شامل مشخصه های دیگری نیز باشد مانند:

```
<?XML Version = "1.0"?>
<XSD:Schema xmlns:xsd='http://www.W3.org/2001/xmlschema'>
...
...
</XSD:Schema>
```

از آنجایی که فایل های XSD خود یک سند XML می باشند در خط اول اعلان XML را قرار داده ایم و در خط دوم به منظور مشخص کردن روش توصیف فایل XML (در اینجا XSD است) نام namespace مرجع مورد نظر را مشخص می کنیم. از این پس هر element ی که پیشوند XSD داشته باشد به این namespace نسبت داده می شود.

element های یک فایل xsd دارای چهار مشخصه اصلی می باشند. به نمونه زیر توجه کنید:

```
<?XML Version = "1.0"?>
<XSD:Schema xmlns:xsd='http://www.W3.org/2001/xmlschema'>
 <XSD:Element Name="student" Minoccurs="0" MaxOccurs="Unbounded"/>
  <XSD:ComplexType>
   <XSD:Sequence>
    <XSD:Element Name="ID" Type="XSD:String" MinOccurs="1" MaxOccurs="1"/>
    <XSD:Element Name="GPA" Type="XSD:String" MinOccurs="1" MaxOccurs="1"/>
   </XSD:Sequence>
  </XSD:ComplexType>
 </XSD:Element>
</XSD:Schema>
```

- Name: **نام element مورد نظر را مشخص می کند.**
- Type: **نوع داده هر element را مشخص می کند.**

- MinOccurs: اجباری یا اختیاری بودن element را مشخص می کند. به صورت پیش فرض عدد یک برای آن در نظر گرفته می شود.
- MaxOccurs: تعداد مجاز هر عنصر را در سند مشخص می کند. به صورت پیش فرض عدد یک برای آن در نظر گرفته می شود. مقدار Unbounded محدودیتی در ایجاد element ایجاد نمی کند.

از تگ <XSD:Sequence/> زمانی استفاده می شود که توالی element ها برای ما مهم باشد. از این تگ فقط در element های مرکب (که بیش از یک عضو دارند) استفاده می شود.

با توجه به مطالبی که گفته شد سند فوق را به صورت زیر آنالیز میکنیم:

این سند دارای element هایی به نام Student می باشد. که می تواند اصلا در سند وجود نداشته باشد و یا به هر تعدادی تعریف شود. student element دارای دو زیر مجموعه (Child) به نام های ID و GPA می باشد. که هر دو زیر مجموعه از نوع String هستند.هر کدام فقط می توانند یک بار در سند ظاهر شوند. و بایستی پشت سر هم بی آیند زیرا از تگ Sequence استفاده شده است.

توصیف داده ها در یک سند XML با استفاده از تکنولوژی XSD ، شامل دو نوع ساده و مرکب می باشد، که برای مشخص کردن نوع داده ساده از تگ <SimpleType> و نوع داده مرکب از تگ <ComplexType> استفاده می شود . لازم است جایگاه استفاده از هر یک را به درستی بدانید. نوع داده مرکب (ComplexType) زمانی استفاده می شود که element مورد نظر دارای زیر مجموعه و یا attribute باشد. و نوع داده ساده (SimpleType) زمانی استفاده می شود که element مورد نظر دارای زیر مجموعه و یا attribute نباشد و یا بخواهیم با اعمال شرایط و محدودیت هایی داده را کنترل کرده و نوع داده جدیدی ایجاد کنیم.

همان طور که در مثال می بینید student element دارای دو زیر مجموعه می باشد پس زیر مجموعه ها را در بین تگ باز و بسته <ComplexType> قرار داده ایم.

حال اگر بخواهیم بر روی GPA element شرطی را اعمال کنیم به طوری که طول آن بیش از چهار حرف نباشد و فرمت آن به صورت (عدد ، عدد ۱ و عدد۲) باشد نیاز به تعریف نوع داده جدید خواهیم داشت:

```
<?XML Version = "1.0"?>
<XSD:Schema xmlns:xsd='http://www.W3.org/2001/xmlschema'>
  <XSD:Element Name="student" Minoccurs="0" MaxOccurs="Unbounded">
    <XSD:ComplexType>
      <XSD:Sequence>
        <XSD:Element Name="ID" Type="XSD:String" MinOccurs="1"
        MaxOccurs="1"/>
        <XSD:Element Name="GPA" Type="StudentGpa"/>
      </XSD:Sequence>
    </XSD:ComplexType>
  </XSD:Element>
  <XSD:simpletype  name="studentgpa" minoccurs="1" maxoccurs="1">
    <XSD:restriction  base="XSD:string">
      <XSD:length value="4"/>
      <XSD:pattern value="\d{1}.\d{2}"/>
    </XSD:restriction>
```

```
</XSD:simpletype>
```

و یا :

```
<?XML Version = "1.0"?>
<XSD:Schema xmlns:xsd='http://www.W3.org/2001/xmlschema'>
  <XSD:Element Name="student" Minoccurs="0" MaxOccurs="Unbounded">
    <XSD:ComplexType>
      <XSD:Sequence>
        <XSD:Element Name="ID" Type="XSD:String" MinOccurs="1"
        MaxOccurs="1"/>
        <XSD:Element Name="GPA" Type="string" Minoccurs="1"
        Maxoccurs="1"/>
        <XSD:simpletype>
          <XSD:restriction  base="XSD:string">
            <XSD:length  value="4"/>
            <XSD:pattern value="\d{1}.\d{2}"/>
          </XSD:restriction>
        </XSD:simpletype>
      </XSD:Sequence>
    </XSD:ComplexType>
  </XSD:Element>
</XSD:Schema>
```

تفاوت دو حالت فوق در امکان استفاده مجدد از نوع داده تعریف شده می باشد. به این صورت که در حالت اول نوع داده studentgpa در تمام سند قابل استفاده خواهد بود اما در نمونه دوم فقط داخل GPA element استفاده خواهد شد و برای element های دیگر نمی توان از آن استفاده کرد.

در تشریح ساختار داده های یک Entity (موجودیت)، برخی مواقع به اجزاء و یا المانها یی برخورد خواهیم کرد که ارتباط تنگاتنگی با Entity دارند. مقادیر در ساختمان داده Entity می توانند به صورت پیش فرض، مقادیر ثابت یا تغییر ناپذیری باشند. به عنوان مثال فرض کنید در یک سازمان بزرگ برای تشکیل پرونده پرسنل، از یک فیلد اطلاعاتی با نام ملیت استفاده می شود.بیش از ۹۸ درصد پرسنل شرکت مزبور دارای ملیت ایرانی بوده و تنها ممکن است یک و یا دو درصد از پرسنل، ملیت غیر ایرانی داشته باشند. با توجه به وضعیت فوق، می توان فیلد ملیت را به صورت پیش فرض مقدار دهی و از تکرار آن در سند XML مربوطه جلوگیری و صرفا ملیت افراد غیر ایرانی را در سند مشخص نمود. در چنین مواردی علاوه بر حفظ یکپارچگی اطلاعات به مقدار زیادی در حجم اطلاعات سند XML نیز صرفه جوئی خواهد شد. در تکنولوژی XSD برای معرفی اینگونه اجزاء، نشانه ای تحت عنوان <Attribute> پیش بینی شده است. گرامر استفاده از attribute به‌صورت زیر است:

```
<XSD:attribute Name="name" Type="simple type" Use="how used" Value="value"/>
```

Name: نام attribute را مشخص می کند.

Type: نوع داده attribute را مشخص می کند.

Use: نحوه استفاده از attribute را مشخص و می تواند مقادیر زیر را داشته باشد:

- Required: اجباری است.
- Default: دارای مقدار پیش فرض است.
- Fixed: درای مقدار ثابت و غیر قابل تغییر است.

- Optional: اختیاری است.
- Prohibited: محافظت شده است.

Value: در صورتی که Use مقدار Fixed یا Default داشته باشد ارزش آن را در Value مشخص می کنیم.

همان طور که قبلا گفته شد برای تعریف یک attribute باید از <ComplexType> استفاده می شود. به مثال زیر توجه کنید:

```
<?XML Version = "1.0"?>
<?XML Version = "1.0"?>
<XSD:Schema xmlns:xsd='http://www.W3.org/2001/xmlschema'>
<XSD:Element Name="student" Minoccurs="0" MaxOccurs="Unbounded">
   <XSD:ComplexType>
    <XSD:Element Name="GPA" Type="XSD:String" MinOccurs="1" MaxOccurs="1"/>
   <XSD:Attribute Name="ID" Type="XSD:String" Use="Required"/>
  </XSD:ComplexType>
</XSD:Element>
</XSD:Schema>
```

حال فرض کنید که در مثال قبل ، قصد داریم بر روی نوع داده attribute ID ، محدودیتی خاصی را اعمال نمائیم "طول رشته پنج و کاراکترهای آن عدد باشند " ، در چنین حالتی توصیف سند بصورت زیر خواهد بود:

```
<?XML Version = "1.0"?>
<XSD:Schema        xmlns:xsd='http://www.W3.org/2001/xmlschema'>
  <XSD:Element Name="student" Minoccurs="0" MaxOccurs="Unbounded">
    <XSD:ComplexType>
      <XSD:Element Name="GPA" Type="XSD:String"MinOccurs="1"MaxOccurs="1"/>
      <XSD:Attribute Name="ID" Use="Required">
        <XSD:simpletype>
          <XSD:restriction  Base="XSD:String">
            <XSD:Length="5" />
            <XSD:Pattern = "\d{5}" />
          </XSD:restriction>
        </XSD:simpletype>
        </XSD:attribute>
      </XSD:ComplexType>
  </XSD:Element>
</XSD:Schema>
```

همان طور که گفته شد امکان استفاده مجدد از یک سند و element های تعریف شده در زیر مجموعه یک Schema وجود دارد، برای این منظور از import یا include استفاده می شود. مثال زیر نحوه استفاده از آن را نشان می دهد:

```
<?XML Version = "1.0"?>
<XSD:Schema Xmlns:XSD="http://www.w3.org/2001/xmlschema"
TargetNameSpace="URI goes here">
<XSD:Include Schemalocation="XSDfilename goes here"/>
      .
      .
      .
</XSD:Schema>
```

زمانی که فایل جاری و فایلی که قصد استفاده آن را داریم (TargetNameSpace) در یک مکان باشد با روش فوق عمل می کنیم اما اگر در location های متفاوتی باشند از دستور import استفاده می شود:

```
<?XML Version = "1.0"?>
<XSD:Schema Xmlns:XSD="http://www.w3.org/2001/xmlschema"
TargetNameSpace="URI goes here">
  <XSD:Import Namespace="URI goes here" Schemalocation = "XSD filename goes
here"/>
  .
  .
  .
</XSD:Schema>
```

تکنولوژی XSD از چهل و چهار نوع DataType ساده یا استاندارد حمایت می کند که بسیاری از آنان از پیش تعریف شده است و برخی دیگر نیز از تعاریف فوق، مشتق شده اند. مثال:

```
<XSD:Element Name="ID" Type="XSD:String" MinOccurs="1" MaxOccurs="1"/>
```

همان گونه که در مثال های قبل ملاحظه کردید برای محدود کردن داده های ورودی می توان در فایل XSD از Regular Expression استفاده کرد. به مثال زیر توجه کنید:

```
<XSD:simpleType name="StudentGPA" minOccurs="1" maxOccurs="1">
  <XSD:restriction base="XSD:string">
    <XSD:length value="4" />
    <XSD:pattern value="/d{1} . /d{2}" />
  </XSD:restriction>
</XSD:simpleType>
```

و در مورد داده های عددی مثال زیر را بررسی نمایید:

```
<XSD:simpleType name="courseNumber" minOccurs="0" maxOccurs="10">
  <XSD:restriction base="XSD:integer">
    <XSD:minInclusive value="1000" />
    <XSD:maxInclusive value="3000" />
  </XSD:restriction>
</XSD:simpleType>
```

در این مثال نوع داده دلخواهی جهت توصیف عنصر شماره درس ایجاد شده است که طبق آن هر دانشجو می تواند تا ۱۰ درس را ثبت نام نموده و شماره هر درس باید بین ۱۰۰۰ تا ۳۰۰۰ باشد. بنابـر این در صورتی که در یک سند XML که از توصیف فوق استفاده نموده وعنصر course دارای مقدار ۴۰۰۰ باشد، پارسر XSD خطائی مبنی بر ایجاد مشکل در نوع داده ارائه می دهد و سند اعتبار سنجی نخواهد شد.

جهت لینک کردن فایل XML به پیش نویس آن ریشه اصلی فایل را به شکل زیر تغییر میدهیم:

```
<?xml version="1.0"?>
<note xmlns="http://www.ostovarit.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ostovarit.com note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
```

```
</note>
```

حال شما می توانید برای اسناد XML فایل XSD ایجاد و یا فایل های XSD موجود را درک و ویرایش نمایید. برای کسب اطلاعات بیشتر و مطالعه عمیق تر این مبحث می توانید به لینک های معرفی شده در نرم افزار رجوع نمایید.

بخش چهارم

زبان های تعریف سَبک (StyleSheet)

XML بین دو مقوله داده، و نمایش داده ها تفکیک قائل شده است. تگ هایی برای نمایش اطلاعات تعریف نشده، به همین دلیل باید از تکنولوژی های دیگر جهت نمایش داده ها استفاده کرد. اطلاعات مربوط به نحوه نمایش داده های فایل XML را در فایلی مجزا ذخیره کرده و با استفاده از یک پردازنده Stylesheet فایل فوق را با سند XML ترکیب می کنیم. استفاده مکرر از یک StyleSheet در پروژه های متفاوت از فواید آن به شمار می آید.

CSS و XSL دو زبان نگراش Stylesheet برای XML هستند.

CSS

یک زبان Style بوده که به منظور استفاده در سند های Html ابداع گردیده و به خوبی قادر به فعالیت روی سندهای XML نیز می باشد. برای استفاده از این زبان باید سند XML را به Stylesheet مورد نظر لینک نماییم مانند:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
.
.
.
</CATALOG>
```

همانطور که ملاحظه میکنید سند فوق به فایل cd_catalog.css اشاره دارد محتویات آن به صورت زیر می باشد:

```
CATALOG
{
background-color: #ffffff;
width: 100%;
```

```
}
CD
{
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE
{
color: #FF0000;
font-size: 20pt;
}
ARTIST
{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
display: block;
color: #000000;
margin-left: 20pt;
}
```

با وجود اینکه با CSS می توان روی فایل های XML فعالیت داشت توصیه می شود جهت فرم دهی به فایل های XML از XSL استفاده شود زیرا این تکنولوژی با XML سازگاری بیشتری دارد و محدودیت های CSS را نخواهد داشت. یادگیری CSS بسیار آسان می باشد اما به دلیل آنکه در اینجا ما را از موضوع اصلی دور می کند از پرداختن به آن خودداری می کنیم در صورت تمایل برای یادگیری این زبان استایل به لینک های معرفی شده در نرم افزار رجوع شود.

XSL

زبان دیگری برای نمایش فایل های XML می باشد که محدودیت های موجود CSS را ندارد. از جمله توانایی های این تکنولوژی می توان به ساخت صفحات HTML قبل از نمایش آنها اشاره داشت. فایل زیر، کد xsl ی را نشان می دهد که eleman <title> را در زمان ایجاد تگهای HTML درشت جلوه می دهد.

```
<xsl:template match="title">
  <H1>
    <xsl:apply-templates/>
  </H1>
</xsl:template>
```

XSL فراتر از یک زبان Stylesheet معمولی است. XSL شامل سه قسمت اصلی می باشد:

- XSLT : زبانی برای تغییر شکل فایل XML
- XPATH : زبانی برای هدایت و اشاره به قسمتی از یک فایل XML
- XSL-FO : زبانی برای قالب بندی یک فایل XML

The World Wide Web Consortium (W3C) started to develop XSL because there was a need for an XML-based Stylesheet Language. XML does not use predefined tags (we can use any tag-names we like), and therefore the meaning of each tag is **not well understood**. A <table> tag could mean an HTML table, a piece of furniture, or something else - and a browser **does not know how to display it**. XSL describes how the XML document should be displayed!

در ادامه به توضیح هر بخش از XSL خواهیم پرداخت:

XSLT (Transformation)

یکی از مهمترین قسمت های XSL می باشد که امکان تغییر یک سند XML را به فرمتی دیگر فراهم می سازد. فرمت جدید میتوانید یک سند XML دیگر یا یک سند XHTML و یا هر فرمت قابل تشخیص توسط مرورگر ها باشد. تمامی مرورگر های موجود XML و XSLT را پشتیبانی می کنند. به وسیله XSLT می توانید یک element را حذف ویرایش و یا اضافه کنید، همچنین با بررسی شرط ها نمایش و یا عدم نمایش یک element را مشخص نمایید و بسیاری توانایی های دیگر.

Element زیر تعریف می کند این سند از یک XSLT پیروی می کند به صورت زیر می باشد:

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

یا:

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

برای مثال فایل XML زیر را در نظر بگیرید:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="cdcatalog.xsl"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
  .
  .
</catalog>
```

خط زرد رنگ اتصال سند XML به فایل XSLT زیر می باشد که با نام cdcatalog.xsl ذخیره شده است:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

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

  <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
        <table border="1">
          <tr bgcolor="#9acd32">
            <th>Title</th>
            <th>Artist</th>
          </tr>
          <xsl:for-each select="catalog/cd">
```

```
            <tr>
              <td>
                <xsl:value-of select="title"/>
              </td>
              <td>
                <xsl:value-of select="artist"/>
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

XSLT is a language for transforming XML documents into XHTML documents or to other XML documents. A common way to describe the transformation process is to say that **XSLT transforms an XML source-tree into an XML result-tree**. XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

در خط اول مثال فوق اعلان XML به چشم می خورد که نشان می دهد فایل های XSLT خود از نوع XML می باشند. خط بعد عنوان استاندارد XSLT است، که در ادامه namespace را با استفاده از خصلت xmlns:xsl تعریف کرده ایم.

عبارت <xsl:template match="/"> یک عبارت مبتنی بر XPath بوده و المان ریشه سند XML را مشخص می نماید. یک سند XML دارای یک ساختار سلسله مراتبی است و همانند یونیکس که از "/" برای مشخص نمودن ریشه استفاده می نماید، XPath نیز از "/" برای مشخص نمودن المان ریشه در یک سند XML استفاده می نماید. در مدل DOM به عنصر فوق "Document object" و در XPath به آن ریشه گفته می شود.

پس از آن بدنه ی Template سند را مشاهده می کنید که از تگ های HTML و متن خروجی تشکیل شده است. مقدار <xsl:value-of> یک دستورالعمل XSLT است و مقدار Node موجود در سند مبدا را در سند خروجی تکثیر می کند. مقدار select نام Node ی که داده ی آن باید انتخاب شود را مشخص می کند.

همان گونه که ملاحظه می کنید <xsl:value-of> را داخل یک <xsl:for-each> آورده ایم، در این حالت کلیه Node های مشخص شده در select خوانده می شود.

برای مرتب سازی داده ها کافیست sort را به کد خود اضافه نمایید:

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

برای شرط گذاری بر روی داده ها از دستور <xsl:if test= استفاده می شود:

```
<xsl:if test="expression">
  ...some output if the expression is true...

</xsl:if>
```

با توجه به مثال اصلی برای شرط گذاری روی داده ها به صورت زیر عمل می کنیم:

```
<xsl:if test="price &gt; 10">
  <tr>
    <td>
      <xsl:value-of select="title"/>
    </td>
    <td>
      <xsl:value-of select="artist"/>
    </td>
  </tr>

</xsl:if>
```

همان طور که در فصل های گذشته گفته شد برای استفاده از کاراکتر هایی که دارای معنای از پیش تعریف شده در XML هستند از معادل آنها استفاده می شود مقدار > معادل است با >.

دستور <xsl:choose> برای بررسی چند شرط متوالی استفاده می شود که به صورت زیر می باشد:

```
<xsl:choose>
  <xsl:when test="expression">
    ... some output ...
  </xsl:when>
  <xsl:otherwise>
    ... some output ....
  </xsl:otherwise>

</xsl:choose>
```

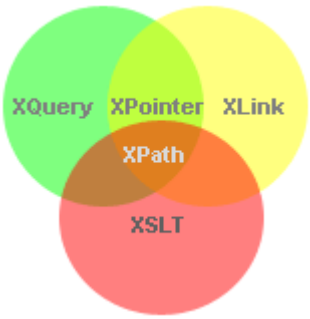
برای مثال:

```
<xsl:choose>
  <xsl:when test="price &gt; 10">
    <td bgcolor="#ff00ff">
      <xsl:value-of select="artist"/>
    </td>
  </xsl:when>
  <xsl:otherwise>
    <td>
      <xsl:value-of select="artist"/>
    </td>
  </xsl:otherwise>

</xsl:choose>
```

XPATH

توسط این ویژگی می توان به قسمتی از یک سند XML اشاره کرد این خاصیت اساس پردازش Stylesheet می باشد. به عبارتی دیگر XPATH زبانی برای استخراج از ساختار XML است. دستور XPATH در XML مانند دستور Select با SQL است. XPATH سازگاری های لازم با XSLT، XPointer و دیگر تکنولوژی های مرتبط با XML را دارا می باشد؛ به طوری که بخش عمده ای از XSLT را تشکیل می دهد و بدون دانش XPATH آزادی کافی برای کار با فایل های XSLT را نخواهید داشت.



برای روشن شدن کاربرد XPath فایل XML زیر را در نظر گرفته و توضیحات را مطالعه فرمایید:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<bookstore>

  <book>
    <title lang="eng">Harry Potter</title>
    <price>29.99</price>
  </book>

  <book>
    <title lang="eng">Learning XML</title>
    <price>39.95</price>
  </book>

</bookstore>
```

برای انتخاب یک نود در سند XML از Path Expression استفاده می شود. جدول زیر مفیدترین Path Expression ها را نشان می دهد:

Expression	توضیحات
نام نود	برای انتخاب همه نودهای فرزند مربوط به نود نام برده شده
/	انتخاب از نود ریشه
//	انتخاب نودها درون سند از روی نود جاری
.	انتخاب نود جاری
..	انتخاب نود والد نود جاری
@	انتخاب صفت ها

گزاره ها (Predicate): برای پیدا کردن یک نود خاص یا یک نود با مقداری مشخص استفاده می شود. Predicate ها همیشه درون علامت [ ] جاسازی می شوند. در جدول زیر لیستی از Path Expression ها به همراه Predicate ها و نتیجه آنها نمایش داده شده است:

Path Expression	Result
/bookstore/book[1]	اولین عنصر book را که فرزند عنصر bookstore می باشد، انتخاب می کند
/bookstore/book[last()]	آخرین عنصر book را که فرزند عنصر bookstore می باشد، انتخاب می کند
/bookstore/book[position()<3]	۲ عنصر book اول را که فرزند عنصر bookstore می باشند، انتخاب می کند
//title[@lang]	همه عنصرهای title را که شامل صفت lang هستند، انتخاب می کند
//title[@lang='eng']	همه عنصرهای title را که شامل صفت lang با مقدار eng هستند را انتخاب می کند
/bookstore/book[price>35.00]	همه عنصرهای book را که فرزند bookstore بوده و مقدار عنصر price در آنها بزرگتر از ۳۵ بوده را انتخاب می کند

انتخاب نودهای نامعلوم (Unknown Node) : با کمک wildcard ها در XPath می توان عناصر ناشناس در اسناد XML را انتخاب کرد.

Wildcard	توضیحات
*	هر عنصری را نمایش می دهد
@*	هر صفتی را نمایش می دهد
Node()	هر نودی (عنصر یا صفت) را نمایش می دهد

جدول زیر مثالهایی را برای روشن کردن این مطلب نمایش می دهد.

Path Expression	Result
/bookstore/*	همه نودهای فرزند عنصر bookstore را انتخاب می کند
//*	همه عنصرهای یک سند را انتخاب می کند
//title[@*]	همه عنصرهای title با هر صفتی که دارند، را انتخاب می کند

برای انتخاب چندین مسیر از عمگر | در یک XPath Expression می توان چندین Path را انتخاب کرد. در جدول زیر مثال هایی از انتخاب چندین مسیر نشان داده شده است.

Path Expression	Result
//book/title | //book/price	همه عنصرهای title و price ، متعلق به عنصر های book را انتخاب می کند
//title | //price	همه عنصرهای title و price را در سند انتخاب می کند

یک Axes مجموعه ای از node های مرتبط با node جاری را تعریف می کند. جدول زیر بعضی از این Axes ها را نشان می دهد.

AxesName	Result
Ancestor	همه اجداد نود جاری را انتخاب می کند.
Ancestor-or-self	همه اجداد نود جاری و همچنین خود نود را انتخاب می کند.
Attribute	همه صفت های نود جاری را انتخاب می کند.

Child	همه فرزندان نود جاری را انتخاب می کند.
Descendant	همه نوه های نود جاری را انتخاب می کند.
Descendant-or-self	همه نوه های نود جاری و خود نود را انتخاب می کند.
Following	هر چیزی در سند که بعد از پایان تگ جاری آمده است را انتخاب می کند.
Following-sibling	همه sibling های بعد از نود جاری را انتخاب می کند.
Namespace	همه نودهای namespace نود جاری را انتخاب می کند.
Parent	والد نود جاری را انتخاب می کند.
Preceding	هر چیزی در سند که قبل از تگ آغاز نود جاری آمده است را انتخاب می کند.
Preceding-sibling	همه sibling های قبل از نود جاری را امنتخاب می کند.
Self	خود نود جاری را انتخاب می کند.

یک Location Path Expression به مسیر نود و یا نودهایی اشاره می کند که این مسیر می تواند بصورت مطلق و یا نسبی تعریف شود . Location path مطلق با / شروع می شود در حالی که location path نسبی نیازی یه این علامت برای شروع ندارد. با این وجود هر یک از این دو نوع شامل گام هایی می باشند، که برای جداسازی آنها از / استفاده می شود.

برای مثال:

An absolute location path:

/step/step/...

A relative location path:

step/step/...

ساختار آن به صورت زیر می باشد:

axisname::nodetest[predicate]

با توجه به ساختار فوق مثال زیر را مطالعه فرمایید:

Example	Result
child::book	همه نود های book که فرزند نود جاری هستند را انتخاب می کند.
attribute::lang	صفت lang نود جاری را انتخاب می کند.
child::*	همه فرزندان نود جاری را انتخاب می کند.
Attribute::*	همه صفت های نود جاری را انتخاب می کند.
child::text()	همه نود های متنی فرزند نود جاری را انتخاب می کند.
child::node()	همه نود های فرزند نود جاری را انتخاب می کند.
descendant::book	همه نوه های book نود جاری را انتخاب می کند.
ancestor::book	همه اجداد book نود جاری را انتخاب می کند.
ancestor-or-self::book	همه اجداد book جاری و خود نود در صورتی که از نوع book باشد.
child::*/child::price	همه نوه های price مربوط به تود جاری را انتخاب می کند.

XSL-FO

زبانی است برای قالب بندی یک فایل XML. XSL-FO گاها همان XSL نامیده میشود. فایلهای آن با پسوند fo. و یا fob. ذخیره می شوند. اما شما می توانید فایل های XSL-FO را با پسوند xml. نیز ذخیره کنید تا برای ویرایشگر های XML قابل فهم و اجرا باشد. نمونه ای از یک فایل را در زیر می بینید:

```
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <!-- The full XSL-FO document goes here -->
  <fo:layout-master-set>
    <!-- All page templates go here -->
  </fo:layout-master-set>
```

با کمک این زبان می توان قوانینی مشابه " تمام عناوین به صورت پر رنگ، با فونتی مشخص، وسط چین، و دارای کادر" را به سادگی تعریف نمود. در این راستا الزامی به نوشتن قوانین متعدد برای هر یک از دستگاه هایی که قصد حمایت از آنان را داریم نخواهیم داشت.

جهت مطالعه بیشتر این تکنولوژی به لینک های معرفی شده در نرم افزار مراجعه نمایید.

بخش پنجم

پردازش اسناد XML

Parser

هر پردازنده XML (مرورگرها، پردازنده های Schema، ویرایشگرها ، پردازنده های Stylesheet) دارای یک پارسر می باشند. پارسر یک سند XML را خوانده و آن را به بخش های متفاوت تقسیم و در حافظه مستقر می نماید (معیار تقسیم بندی می تواند بر اساس المان و یا خصلت باشد). پس از استقرار بخش های متفاوت یک سند XML در حافظه، امکان انجام عملیات بر روی هر یک از بخش های موجود توسط پردازنده فراهم می گردد. (نظیر داده های موجود در یک بانک اطلاعاتی):

- انتقال یک سند از یک فرمت به فرمت دیگر
- بازسازی مجدد یک سند XML با توجه به اولویت های اطلاعاتی مورد نظر
- اعمال فرمت های دلخواه به‌منظور چاپ و یا نمایش

اکثر پیاده کنندگان نرم افزار های مبتنی بر XML، ضرورتی به فراگیری جزئیات مربوط به نحوه پارسینگ، نداشته و می توانند آن را به ابزارهای ارائه شده توسط تولیدکنندگان واگذار نمایند. (در محیط های پیاده سازی امکانات مربوطه در این زمینه پیش بینی می گردد . با مطالعه پارسرهای XML، شناخت مناسبی نسبت به امکانات و ویژگی های موجود در آنها ایجاد خواهد شد. XML دارای دو اینترفیس برنامه نویسی DOM و SAX است.

(DOM) Document Object Model

برخلاف نام خود یک مدل واقعی نیست ، بلکه یک API (اینترفیس برنامه نویسی) برای نوشتن کدهای لازم ب ه منظور انجام عملیات دلخواه، بر روی بخشی از یک سند XML، پس از استقرار آن در حافظه است. مهمترین مزیت پارسرهای سازگار با DOM، ارائه استانداردهای لازم API به‌منظور انجام عملیات دلخواه در رابطه با اطلاعات است.

The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them. In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

(SAX) Simple API for XML

SAX بر اساس یک مدل شی گراء رفتار نمی نماید. در این اینترفیس ، از مجموعه ای از رویدادها (Event) استفاده می گردد. رویدادها، پس از خواندن یک سند XML توسط پارسرهای سازگار با SAX، با توجه به شرایط مربوطه فعال خواهند شد. یکی از دلایل مطرح شدن SAX با توجه به وجود DOM، عدم امکان استفاده از دستورات DOM تا زمانی‌ست که تمام سند در حافظه مستقر شود. استفاده و بکارگیری از دستورات DOM صرفاً پس از استقرار تمام

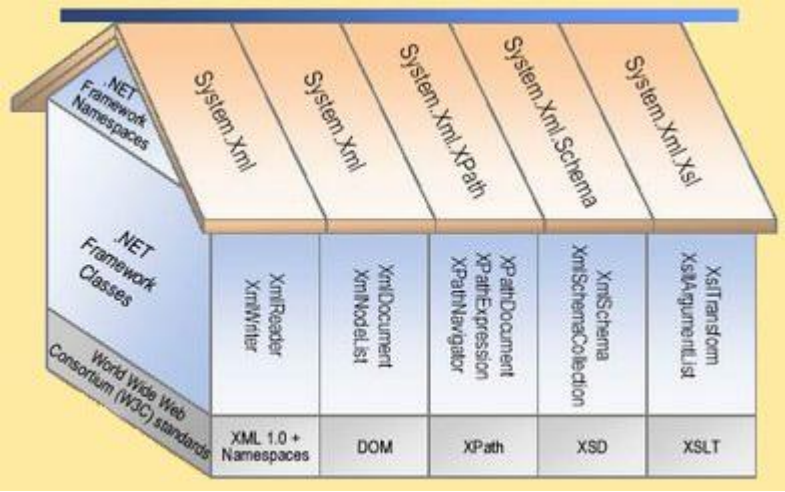
سند XML در حافظه امکان پذیر خواهد بود . بدیهی است با توجه به رویکرد عملیاتی فوق ، حجم بالای سندهای XML می تواند باعث بروز مسائل و مشکلاتی در این رابطه گردد. مثلاً ب ه منظور دستیابی به یک المان خاص می بایست در انتظار استقرار تمام سند XML در حافظه گردید. (از دست دادن زمان و مصرف بیهوده حافظه). با استفاده از تکنولوژی SAX می توان به‌سرعت یک سند را فعال و در ادامه المان های موجود را به‌منظور استفاده از محتویات مورد نظر جستجو کرد.

قابلیت SAX، یک پیش نیاز برای پارسرهای سازگار با DOM است، چون یک پارسر DOM می بایست در ابتدا سند را خوانده و در ادامه از رویدادهای مشابه SAX به‌منظور تشخیص المان ها استفاده و آنها را در حافظه ایجاد نماید. به‌عنوان جمع بندی می بایست به این نکته اشاره گردد که اغلب پارسرها امکانات مربوط به DOM و SAX را حمایت کرده و می توان از پارسر های مشابه به‌منظور انجام هر یک از حالات پردازش استفاده کرد.

بخش ششم

XML در NET.

XML در دات نت دارای نقشی بسیار مهم و محوری بوده و لازم است با جایگاه آن به صورت اصولی آشنا شویم. XML، یک تکنولوژی استاندارد ایده آل برای برنامه هایی است که بر روی بستر اینترنت اجراء می گردند. ماکروسافت در پروژه دات نت، از استانداردهای کنسرسیوم وب پیروی و نسخه های اختصاصی خود را طراحی و پیاده سازی نموده است. بدین ترتیب (با توجه به تبعیت ماکروسافت از استانداردهای کنسرسیوم وب) می توان این اطمینان را بدست آورد که نرم افزارهای تولید شده در دات نت، قابلیت ارتباط با سایر برنامه های تولید شده مبتنی بر استانداردهای کنسرسیوم وب را بخوبی دارا می باشند. در محیط اینترنت، داده ها می توانند از منابع متفاوت و به اشکال گوناگون دریافت گردند. سرویس های وب XML و سایر برنامه هایی که با استفاده از دات نت پیاده سازی می گردند، مسائل و مشکلات مربوط به انجام عملیات بر روی داده هایی با فرمت های متفاوت و از منابع گوناگون، را برطرف می نماید. محیط دات نت، شامل مجموعه ای از محصولات است که بر اساس XML و سایر استانداردهای اینترنت، ایجاد شده اند. محیط فوق، برای هر یک از جنبه های مرتبط با پیاده سازی، مدیریت، استفاده، سرویس های وب XML، امکانات و راهکارهای مناسبی را ارائه داده است. سرویس های وب XML، این امکان را به برنامه ها خواهند داد تا قادر به اشتراک اطلاعات از طریق اینترنت صرفنظر از سیستم عامل و زبان برنامه نویسی مربوطه باشند. با استفاده از XML در دات نت، می توان اغلب مشکلات مربوط به پیاده سازی نرم افزار بر روی اینترنت را که در حال حاضر با آن مواجه هستیم، برطرف کرد. XML، یک راه حل جامع برای تشریح و مبادله داده های ساخت یافته را ارائه می نماید.



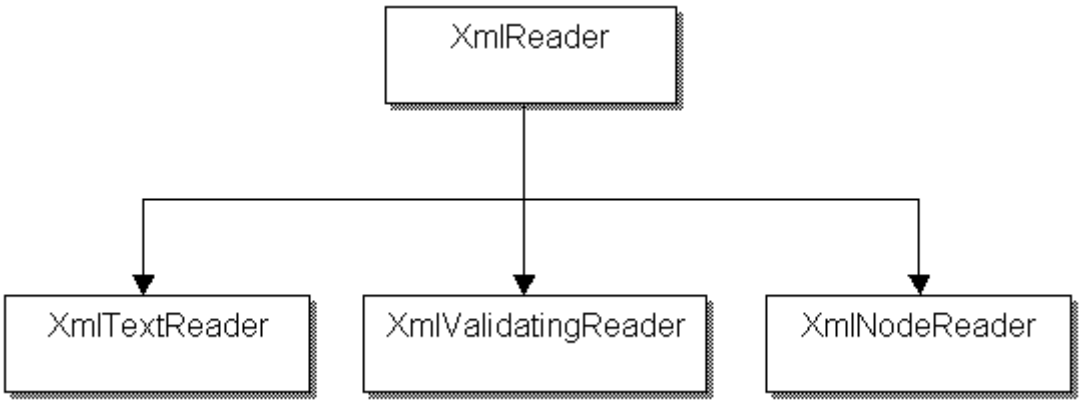
Using **MSXML**(Microsoft XML Core Services), you can build XML-based applications. You can easily use MSXML in Visual Studio 6.0 applications as DOM and SAX Parser with support for XSLT and XPath. The **System.Xml** namespace provides standards-based support for processing XML. System.xml is not just the managed version of MSXML; its functionality may overlap that of the MSXML COM library, and it also contains a rich object model and hierarchy.

ذخیره سازی داده ها و انتقال اطلاعات از اهمیت بالایی در فریمورک دات نت محسوب می گردد. بنابراین تمام ملاحظات امنیتی در این فریمورک به رویکردهای امنیتی XML مربوط می گردد. کلاس SecurityElement مدل XML لازم به منظور رمزکردن اشیاء را ارائه می نماید. کلاس فوق، به منظور استفاده همراه یک سیستم ایمنی در نظرگرفته شده است و امکان استفاده از آن به عنوان یک کلاس شی XML عمومی وجود نخواهد داشت.

در مواردی که یک سند XML ایمن شده باشد، بدون ارسال مدارک لازم، امکان دستیابی و استفاده از آن وجود نخواهد داشت. کلاس XmlTextReader، امکان ارسال مدارک را از طریق استفاده از کلاس CredentialCache موجود در System.Net فراهم می نماید.

دات نت پنج namespace را برای پشتیبانی از کلاس های XML دارد که عبارت است از: System.Xml .System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath, System.Xml.Xsl

فضای نام (namespace) System.Xml کلاس های مربوط به خواندن و نوشتن فایل های XML را در خود دارد. کلاس های XmlTextReader، XmlNodeReader، XmlValidatingReader (مشتق شده از XmlReader) کلاس های اصلی برای خواندن می باشند و XmlTextWriter، XmlNodeWriter (مشتق شده از XmlWriter) کلاس های اصلی برای نوشتن یک فایل XML هستند.



XmlTextReader

بعد از ساخت نمونه از این کلاس با استفاده از متد read به راحتی می توانید فایل XML مورد نظر را بخوانید. خواندن فایل از Node اول شروع و تا زمانی که متد مقدار false را برگرداند ادامه دارد.

برای بدست آوردن مشخصات نود می توان از پراپرتی های Name، Depth، Value و ... استفاده کرد. برای مثال بعد از ساخت یک نمونه از کلاس XmlTextReader و فراخوانی متد read پراپرتی های نود خوانده شده را بررسی می کنیم:

```
XmlTextReader reader = new XmlTextReader(@"C:\XMLFile1.xml");
while (reader.Read())
{
  if (reader.HasValue)
```

```
    {
      Console.WriteLine("Name : " + reader.Name);
      Console.WriteLine("Node Depth: " + reader.Depth.ToString());
      Console.WriteLine("Value : " + reader.Value);
    }
}
```

برای تشخیص نوع نود خوانده شده از کلاس XmlNodeType کمک می گیریم:

```
XmlNodeType type = reader.NodeType;
```

مقادیر Attribute، CDATA، Comment، Document، Element، WhiteSpace و... می تواند حاصل پراپرتی NodeType باشد. در فرم ReadXMLtxtBox.cs به وسیله کد زیر ابتدا نوع نود بررسی شده و با توجه به آن در تکست باکس درج گردیده است.

```
XmlTextReader reader = new XmlTextReader(txtRoot.Text);
while (reader.Read())
{
    switch (reader.NodeType)
    {
        case XmlNodeType.Element: // The node is an element.
            txtXMLValues.Text = txtXMLValues.Text + "<" + reader.Name;
            while (reader.MoveToNextAttribute())
            {
                // Read the attributes.
                txtXMLValues.Text = txtXMLValues.Text +" " + reader.Name +
                "='" + reader.Value + "'";
            }
            txtXMLValues.Text = txtXMLValues.Text + ">\n";
            break;
        case XmlNodeType.Text: //Display the text in each element.
            txtXMLValues.Text = txtXMLValues.Text+"    "+reader.Value+"\n";
            break;
        case XmlNodeType.EndElement: //Display the end of the element.
            txtXMLValues.Text = txtXMLValues.Text+"</"+reader.Name+">\n";
            break;
    }
}
```

بعد از بررسی مثال فوق، فرم ReadXMLListBox.cs را مطالعه فرمایید. در این مثال ملاحظه خواهید کرد که چطور می توان یک فایل XML را با توجه به مطالب گفته شده در یک List View به نمایش در آورد.

یکی دیگر از متد های پر کاربرد skip می باشد. در مثال زیر برای یافتن مشخصات نود bookstore مقدار name را برای هر نود بررسی کرده در صورت برابر نبودن، متد skip را فراخوانی کرده، در غیر این صورت مشخصات نود را می خوانیم:

```
while (reader.Read())
{
    // Look for a Node with name bookstore
    if (reader.Name != "bookstore")
        reader.Skip();
    else
```

```
    {
        Console.WriteLine("Name: " + reader.Name);
        Console.WriteLine("Level of the node:" + reader.Depth.ToString());
        Console.WriteLine("Value: " + reader.Value);
    }
}
```

متد هایی مانند MoveToFirstAttribute، MoveToContent، MoveToAttribute، MoveToElement و ... برای حرکت بین نود ها با توجه به نوع آنها ایجاد شده است که می توان در زمان استفاده از فایل های XML برای افزایش سرعت و بهینه سازی برنامه از آنها استفاده کرد. شما می توانید از پراپرتی HasAttributes برای چک کردن وجود یا عدم وجود Attribute در نود جاری استفاده کرده و با استفاده از AttributesCount تعداد آن ها را به دست آورید. برای مثال:

```
using System;
using System.Xml;

class XmlReaderSamp
{
    static void Main(string[] args)
    {
        XmlTextReader reader = new XmlTextReader(@"C:\XMLFile1.xml");
        reader.MoveToContent();
        reader.MoveToFirstAttribute();
        Console.WriteLine("First Attribute value" + reader.Value);
        Console.WriteLine("First Attribute Name" + reader.Name);

        while (reader.Read())
        {
            if (reader.HasAttributes)
            {
                Console.WriteLine(reader.Name + "Attribute");
                for (int i = 0; i < reader.AttributeCount; i++)
                {
                   reader.MoveToAttribute(i);
                    Console.WriteLine("Name: " + reader.Name + ",value: " +
                   reader.Value);
                }

                reader.MoveToElement();
            }
        }
        Console.ReadLine();
    }
}
```

XmlNode

نقش مهمی را در میان کلاس های مربوط به XML در دات نت بازی می کند. همچنین پراپرتی هایی جهت بدست آوردن ریشه اصلی، نام نودها، آخرین نود، نوع نود و ... دارد. سه کلاس XmlDocument ، XmlDataDocument و XmlDocumentFragmen برای حرکت، ویرایش، حذف و فعالیت های دیگر

روی نود ها از آن مشتق می شوند. کلاس XmlDataDocument دارای متد و پراپرتی هایی جهت کار با ADO.NET می باشد.

XmlDocument

با استفاده از این کلاس می توانید یک فایل XML را بارگذاری نمایید، در طول نود های آن حرکت کنید، مقادیر آن را ویرایش و حاصل را ذخیره نمایید.

به فرم CreateXMLsimple.cs توجه فرمایید. ابتدا نمونه ای از کلاس XmlDocument ایجاد، سپس خط تعریف فایل XML به وسیله کلاس XmlDeclaration و متد CreateXmlDeclaration ساخته شد. با استفاده از متد InsertBefore خط اعلان XML را به ابتدای سند اضافه می کنیم.

```
XmlDocument xmldoc = new XmlDocument();

XmlDeclaration xmlDeclaration = xmldoc.CreateXmlDeclaration("1.0", "utf-8",
null);
xmldoc.InsertBefore(xmlDeclaration, xmldoc.DocumentElement);
```

در ادامه ابتدا ریشه اصلی را ساخته سپس element های مورد نظر را به وسیله کلاس XmlElement ایجاد و به ریشه اصلی اضافه می کنیم.

```
XmlElement xmlelem = xmldoc.CreateElement("", "Users", "");
xmldoc.AppendChild(xmlelem);
XmlElement xmlelem2 = xmldoc.CreateElement("", "User", "");
xmldoc.ChildNodes.Item(1).AppendChild(xmlelem2);
. . .
```

برای ایجاد فایل XML از متد save استفاده می کنیم:

```
xmldoc.Save(Application.StartupPath + "\\ostovarit.xml");
```

در فرم ReadXMLTreeView.cs فایل XML انتخاب شده توسط کاربر را در یک کنترل TreeView با استفاده از کلاس XmlDocument به نمایش در می آوریم. برای درک بهتر این مثال مراحل را یک به یک مورد بررسی قرار دهید.

XmlConvert

این کلاس زیر مجموعه ی System.Xml می باشد و وظیفه ی Encodes و Decodes کردن مقادیر را در یک فایل XML بر عهده دارد. در بسیاری از زبان ها امکان استفاده از کاراکتر های Unicode و کاراکتر های خاص وجود دارد، اما در XML غیر مجاز بوده و نیاز به تبدیل دارند. برای مثال استفاده از کلمه 'Order Detail' در

پایگاه داده برای عنوان ستون، مشکلی ایجاد نخواهد کرد اما در XML فاصله بین دو کلمه مجاز نخواهد بود. کلاس XmlConvert از متدهایی برای تبدیل نوع داده (XML Schema Definition(XSD به نوع داده (CLI) Common Language Infrastructure نیز پشتیبانی می کند. در مثال زیر مقادیری را از جنس Boolean و DateTime در یک فایل XML با استفاده از کلاس XMLTextWriter ذخیره می کنیم.

```
using System;
using System.Xml;

class XmlReaderSamp
{
    static void Main(string[] args)
    {
        XmlTextWriter writer = new XmlTextWriter(@"C:\XMLFile1.xml", null);
        writer.WriteStartElement("MyTestElements");
        bool b1 = true;
        writer.WriteElementString("TestBoolean", XmlConvert.ToString(b1));
        DateTime dt = new DateTime(2000, 01, 01);
        writer.WriteElementString("test date", XmlConvert.ToString(dt));
        writer.WriteEndElement();
        writer.Flush();
        writer.Close();
    }
}
```

XmlWriter

همان طور که از نامش مشخص است برای نوشتن و ایجاد فایل های XML کاربرد دارد و کلاس های متعددی از آن مشتق می شود. در فرم XMLWriterSample.cs نحوه ایجاد سند با کلاس XmlWriter و استفاده از XmlConvert نمایش داده شده است. که در ادامه به توضیح آن خواهیم پرداخت.

XmlWriterSettings

همان طور که گفته شد برای ایجاد فایل XML از XmlWriter استفاده می کنیم. اما گاهی نیاز به اعمال تنظیماتی روی فایل نهایی خواهیم داشت. برای این منظور از XmlWriterSettings استفاده می کنیم. به مثال زیر توجه فرمایید:

```
XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true;
settings.OmitXmlDeclaration = true;
settings.NewLineOnAttributes = false;

XmlWriter writer = XmlWriter.Create(Application.StartupPath +
"\\XMLWriter.xml", settings);

writer.WriteStartElement("order");
writer.WriteAttributeString("orderID", txtOrderID.Text.Trim());
writer.WriteAttributeString("date", dateTimePicker1.Text);
if (txtPrice.Text.Trim() != "")
```

```
{
    writer.WriteElementString("price", txtPrice.Text.Trim());
}
else
{
    writer.WriteElementString("price", XmlConvert.ToString(price));
}
    writer.WriteEndElement();

writer.Flush();
```

در مثال فوق تنظیمات مورد نظر را به نمونه XmlWriterSettings نسبت می دهیم (این تنظیمات می تواند برای مثال مرتب کردن element های تو در تو باشد و یا رفتن به خط جدید بعد از اتمام element) سپس از کلاس XmlWriter نمونه ای را ساخته و نمونه XmlWriterSettings را به عنوان یک پارامتر به متد Create می دهیم. در هنگام ایجاد element ها در صورت نیاز تغییر فرمت داده ها را با استفاده از XmlConvert انجام می دهیم.

XmlTextWriter

از XmlWriter مشتق شده و دارای متد و پراپرتی هایی جهت نوشتن فایل های XML می باشد. constructors آن قادر به دریافت سه مقدار string، stream و یا TextWriter می باشد. برای مثال یک نمونه ایجاد می کنیم:

```
// Create a new file in C:\\ dir
XmlTextWriter textWriter = new XmlTextWriter("C:\\myXmFile.xml", null);
```

بعد از ساخت نمونه متد WriterStartDocument برای آغاز سند و WriteEndDocument برای مشخص کردن پایان آن به کار می بریم. این دو متد سند را برای نوشتن باز و بسته می کنند. پس از آن متد Close را می نویسیم.

```
textWriter.WriteStartDocument();
.
.
.
textWriter.WriteEndDocument();
textWriter.Close();
```

متد WriteComment برای نوشتن کامنت در فایل XML استفاده می شود. متد WriteString برای نوشتن استرینگ به کار می رود و متد های WriteStartAttributeو WriteEndAttribute برای ایجاد Attribute ها استفاده می شوند.

در فرم XmlTextWriterSample از متد های متعددی برای ساخت فایل XML استفاده شده است :

```
// Create a new file in C:\\ dir
XmlTextWriter textWriter = new XmlTextWriter("C:\\myXmFile.xml", null);
// Opens the document
textWriter.WriteStartDocument();
```

```
// Write comments
textWriter.WriteComment("First Comment XmlTextWriter Sample Example");
textWriter.WriteComment("myXmlFile.xml in root dir");
// Write first element
textWriter.WriteStartElement("Student");
textWriter.WriteStartElement("r", "RECORD", "urn:record");
// Write next element
textWriter.WriteStartElement("Name", "");
textWriter.WriteString("Student");
textWriter.WriteEndElement();
// Write one more element
textWriter.WriteStartElement("Address", "");
textWriter.WriteString("Colony");
textWriter.WriteEndElement();
// WriteChars
char[] ch = new char[3];
ch[0] = 'a';
ch[1] = 'r';
ch[2] = 'c';
textWriter.WriteStartElement("Char");
textWriter.WriteChars(ch, 0, ch.Length);
textWriter.WriteEndElement();
// Ends the document.
textWriter.WriteEndDocument();
// close writer
textWriter.Close();
```

پس از اجرای کد فوق فایل XML به آدرس C:\\myXmFile.xml ساخته خواهد شد که حاوی مقادیر زیر است:

```
<?xml version="1.0" ?>
<!-- First Comment XmlTextWriter Sample Example -->
<!-- myXmlFile.xml in root dir -->
<Student>
 <r:RECORD xmlns:r="urn:record">
   <Name>Student</Name>
   <Address>Colony</Address>
   <Char>arc</Char>
 </r:RECORD>
</Student>
```

XslTransform

همان طور که در فصل های گذشته توضیح داده شد از XSLT برای تبدیل یک فایل XML به فرمت های دیگر استفاده می شود. در اینجا قصد داریم یک فایل XML را به کمک یک فایل XSL به فرمت HTML در بیاوریم. قبل از بررسی مثال به این نکته توجه فرمایید که در گذشته برای این منظور از کلاس XslTransform استفاده می شد اما در حال حاضر XslCompiledTransform کلاسی ارتقاع یافته از آن می باشد که نه تنها متد ها و پراپرتی های قبلی را در خود حفظ کرده بلکه افزایش سرعت چشم گیری در انجام پروسه ها داشته است. کلاس فوق را می توانید در فضای System.Xml.Xsl پیدا کنید.

فرم TransformXML.cs نمونه ای از مطالبی است که در بالا گفته شده:

```
//load the Xml doc
XmlDocument myXmlDoc = new XmlDocument();
```

```
myXmlDoc.Load(sXmlPath);
//The XSLT architecture has been redesigned in the Microsoft .NET Framework
version 2.0 release. The XslTransform class has been replaced by the
XslCompiledTransform class.
XslCompiledTransform myXslTrans = new XslCompiledTransform();
//load the Xsl
myXslTrans.Load(sXslPath);
//create the output stream
XmlTextWriter myWriter = new XmlTextWriter("result.html", null);
//do the actual transform of Xml
myXslTrans.Transform(myXmlDoc, null, myWriter);
myWriter.Close();
```

در ابتدا فایل XML را با استفاده از کلاس XmlDocument در حافظه بارگذاری کرده سپس نمونه ای از XslCompiledTransform ساخته و با استفاده از متد load فایل XSL را بارگذاری می نماییم. از کلاس XmlTextWriter برای ساخت فایل HTML کمک می گیریم و در نهایت دو نمونه را برای تبدیل نهایی به متد Transform ارسال می کنیم. متد Transform سه نوع ورودی را می تواند دریافت کند، در مواردی که قصد ویرایش فایل XML قبل از انجام عملیات مربوطه را داشته باشیم نمونه ای از XmlDocument ایجاد می کنیم، در مواردی که با Dataset در ارتباط هستیم کلاس XMLDataDocument استفاده می شود ، و زمانی که قصد استفاده از XPath را داریم از کلاس XPathDocument نمونه ای ایجاد می کنیم. از میان سه کلاس نام برده کلاس XPathDocument از سرعت نسبتا بالاتری برخوردار است. در نهایت پس از فراخوانی متد Transform فایل result.html با اطلاعات فایل XML و مشخصه های فایل XSL ایجاد می شود.

XsltSettings

به طور پیش فرض کلاس XslCompiledTransform از اسکریپت های داخل فایل XSL پشتیبانی نمی کند. برای تنظیم و فعالسازی اسکریپت ها از XsltSettings نمونه ای ایجاد کرده و آن را به متد load XslCompiledTransform ارسال می کنیم. فرم TransformXML2.cs نمونه ای از مطالب فوق می باشد. به مثال زیر توجه نمایید:

```
// Create the XsltSettings object with script enabled.
XsltSettings settings = new XsltSettings(false, true);
// Execute the transform.
XslCompiledTransform xslt = new XslCompiledTransform();
xslt.Load("calc.xsl", settings, new XmlUrlResolver());
xslt.Transform("books.xml", "books.html");
```

XmlSchema

همان طوری که در فصول گذشته گفته شد فایل های Schema برای اعتبار سنجی سند XML استفاده می شوند. این فایل ها دارای گرامری قابل فهم و آسان هستند و از نظر ساختار مشابه یک فایل XML می باشند، با این وجود توصیه می شود قبل از بررسی مثال زیر ابتدا بر مفاهیم یک سند Schema مسلط شوید سپس مثال زیر را که نحوه

ساخت یک سند XSD را نمایش می دهد مطالعه نمایید. کلاس XmlSchema به منظور فعالیت بر روی فایل های Schema ایجاد شده است.

```
XmlSchema schema = new XmlSchema();

// <xs:element name="cat" type="xs:string"/>
XmlSchemaElement elementCat = new XmlSchemaElement();
schema.Items.Add(elementCat);
elementCat.Name = "cat";
elementCat.SchemaTypeName = new XmlQualifiedName("string",
"http://www.w3.org/2001/XMLSchema");

// <xs:element name="dog" type="xs:string"/>
XmlSchemaElement elementDog = new XmlSchemaElement();
schema.Items.Add(elementDog);
elementDog.Name = "dog";
elementDog.SchemaTypeName = new XmlQualifiedName("string",
"http://www.w3.org/2001/XMLSchema");

// <xs:element name="redDog" substitutionGroup="dog" />
XmlSchemaElement elementRedDog = new XmlSchemaElement();
schema.Items.Add(elementRedDog);
elementRedDog.Name = "redDog";
elementRedDog.SubstitutionGroup = new XmlQualifiedName("dog");

// <xs:element name="brownDog" substitutionGroup ="dog" />
XmlSchemaElement elementBrownDog = new XmlSchemaElement();
schema.Items.Add(elementBrownDog);
elementBrownDog.Name = "brownDog";
elementBrownDog.SubstitutionGroup = new XmlQualifiedName("dog");


// <xs:element name="pets">
XmlSchemaElement elementPets = new XmlSchemaElement();
schema.Items.Add(elementPets);
elementPets.Name = "pets";

// <xs:complexType>
XmlSchemaComplexType complexType = new XmlSchemaComplexType();
elementPets.SchemaType = complexType;

// <xs:choice minOccurs="0" maxOccurs="unbounded">
XmlSchemaChoice choice = new XmlSchemaChoice();
complexType.Particle = choice;
choice.MinOccurs = 0;
choice.MaxOccursString = "unbounded";

// <xs:element ref="cat"/>
XmlSchemaElement catRef = new XmlSchemaElement();
choice.Items.Add(catRef);
catRef.RefName = new XmlQualifiedName("cat");

// <xs:element ref="dog"/>
XmlSchemaElement dogRef = new XmlSchemaElement();
choice.Items.Add(dogRef);
dogRef.RefName = new XmlQualifiedName("dog");

XmlSchemaSet schemaSet = new XmlSchemaSet();
schemaSet.Add(schema);
schemaSet.Compile();
```

```
XmlSchema compiledSchema = null;

foreach (XmlSchema schema1 in schemaSet.Schemas())
{
    compiledSchema = schema1;
}

XmlNamespaceManager nsmgr = new XmlNamespaceManager(new NameTable());
nsmgr.AddNamespace("xs", "http://www.w3.org/2001/XMLSchema");

XmlWriterSettings settings = new XmlWriterSettings();
settings.Indent = true;
settings.OmitXmlDeclaration = true;
settings.NewLineOnAttributes = false;
XmlWriter xmlr = XmlWriter.Create("C:\\MakeXSD.xsd", settings);
compiledSchema.Write(xmlr, nsmgr);
xmlr.Close();

StreamReader sr = new StreamReader("C:\\MakeXSD.xsd");
richTextBox1.Text = sr.ReadToEnd();
sr.Close();
}
```

همان طور که ملاحظه می کنید کلاس های بی شماری از کلاس XmlSchema مشتق می شود که تمامی نیاز های شما را برای ساخت یک فایل XSD پوشش می دهند. در مثال فوق ابتدا به تعریف element ها و نوع داده ها پرداختیم و ساختار فایل XSD را مشخص کردیم. خطوط کامنت شده در بین کد ها حاصل اجرای کد را نمایش می دهند که در مورد هر یک در فصول گذشته صحبت شده است. از کلاس XmlNamespaceManager برای تعریف Namespace مورد نظر که در اینجا XS است استفاده کردیم. سپس نمونه ای را از کلاس XmlWriter برای ساخت و ذخیره فایل XSD ایجاد می کنیم. همان طور که از پیش گفته شد کلاس XmlWriterSettings به ما امکان اعمال تنظیمات روی فایل مقصد را می دهد. در صورتی که از تنظیمات فوق استفاده نشود کلیه خطوط فایل XSD پشت سر هم و نا منظم خواهد آمد. مثال فوق را می توانید در فرم XmlSchemaSample.cs بیابید.

XmlReaderSettings

یکی از سریع ترین روش های اعتبار سنجی یک سند XML، استفاده از پراپرتی Schema از کلاس XmlReaderSettings برای مشخص کردن فایل XSD و پراپرتی ValidationType برای مشخص کردن نوع اعتبار سنجی می باشد. این کلاس در فضای System.Xml قرار دارد. فرم XmlReaderSettingsSample.cs نمونه ای از مطلب گفته شده را نشان می دهد.

```
try
{
    XmlReaderSettings settings = new XmlReaderSettings();
    settings.Schemas.Add(null, txtXSD.Text);
    settings.ValidationType = ValidationType.Schema;
    XmlDocument document = new XmlDocument();
    document.Load(txtXML.Text);
    XmlReader rdr = XmlReader.Create(new StringReader(document.InnerXml),
    settings);
```

```
    while (rdr.Read()) { }
    MessageBox.Show("valid");
}
catch
{
    MessageBox.Show("Invalid!");
}
```

XmlValidatingReader

در فضای نام System.Xml قرار دارد و برای اعتبار سنجی اسناد به وسیله DTD، Schema و XDR استفاده می شود. فرم XmlValidatingReaderSample.cs مثالی از این کلاس را نمایش می دهد که به بررسی آن می پردازیم.

```
public void ValidateXML(string filename)
{
    isValid = true;
    error = "";
    try
    {
        XmlTextReader xml = new XmlTextReader(filename);
        XmlValidatingReader xsd = new XmlValidatingReader(xml);
        //Validation Type
        if (radioButton1.Checked)
            xsd.ValidationType = ValidationType.Schema;
        else if (radioButton2.Checked)
            xsd.ValidationType = ValidationType.DTD;
        else
            xsd.ValidationType = ValidationType.None;
        //and validation errors events go to...
        xsd.ValidationEventHandler += new
        ValidationEventHandler(MyValidationEventHandler);
        //wait until the read is over, its occuring in a different thread
        while (xsd.Read())
        {
        }
        xsd.Close();
        // Check whether the document is valid or invalid.
        if (isValid)
            MessageBox.Show("Document is valid");
        else
            MessageBox.Show("Document is invalid");
    }
    catch (UnauthorizedAccessException a)
    {
        //dont have access permission
        error = a.Message;
    }
    catch (Exception a)
    {
        //and other things that could go wrong
        error = a.Message;
    }
    resultErrors.Text = error;
}

//handle our validation errors
```

```
public static void MyValidationEventHandler(object
sender,ValidationEventArgs args)
{
    isValid = false;
    error += args.Message + "\n\n";
}
```

در مثال فوق فضاهای زیر مورد استفاده قرار می گیرند:

```
using System.Xml;         // for XmlTextReader and XmlValidatingReader
using System.Xml.Schema; // for XmlSchemaCollection (which is used later)
```

ابتدا نمونه ای از کلاس XmlTextReader برای خواندن فایل XML ایجاد کرده ایم. سپس از کلاس XmlValidatingReader نمونه ای را می سازیم. با کمک پراپرتی ValidationType نوع اعتبار سنجی را تعیین می کنیم. در صورتی که در زمان اعتبار سنجی خطایی رخ دهد رویدادی ایجاد می شود. به وسیله ValidationEventHandler متدی به نام MyValidationEventHandler ایجاد کرده ایم و در زمان رخ دادن رویداد مقدار خطا را دریافت می کنیم و پیغامی مبنی بر اینکه سند XML فاقد اعتبار است را نمایش می دهیم.

XmlSchemaCollection و XmlSchemaSet

در فضای System.Xml.Schema تعریف شده است و می تواند مجموعه ای از Schema ها را در خود جای دهد. گاها با کلاس XmlValidatingReader استفاده می شود. فرم XmlSchemaCollectionSample.cs نمونه ای از کارایی این کلاس را نمایش می دهد. با آمدن نسخه های جدید XML و در راستای ایجاد تغییراتی جهت استاندارد سازی فرایندهای اعتبار سنجی، این کلاس کمتر مورد استفاده قرار گرفته و جای خود را به کلاس XmlSchemaSet داده است. ازتفاوت های مهم این دو کلاس می توان به موارد زیر اشاره کرد:

- در XmlSchemaCollection از XML schemas و XDR پشتیبانی می شود اما در XmlSchemaSet فقط از XML schemas پشتیبانی می شود.
- متـــد Add در XmlSchemaCollection فایل Schema را کامپایل می کند اما در XmlSchemaSet این طور نیست.
- XmlSchemaCollection تـــوانایی دریـــافت چند Schema با namespace هـــای مشابه را ندارد اما این مسئله در XmlSchemaSet بر طرف شده است.

XmlSerializer

فرآیندی برای تبدیل یک آبجکت (یا گرافی متشکل از چند آبجکت) به یک حالت خطی (و جریان وار) از بایت ها جهت انتقال و یا ذخیره سازی در محلی دیگر است.

Deserialize : عمل برعکس Serialize است؛ یعنی دوباره ساختن آبجکت از روی جریانی از بایت ها .

انواع فرمت ها در سریالیزشین :

۱- Binary : به صورت باینری سریالایز می کند. (فشرده ترین و efficient ترین حالت است)

۲ - SOAP : به صورت soap سریالایز می کند. (برای ارسال در شبکه و جایی که از تحت دات نت بودن کلاینت مطمئن نیستیم)

۳ - XML : به صورت xml سریالایز می کند. (برای ارسال تحت شبکه – مزیت : خوانایی)

فضای System.Xml.Serialization کلاس های مربوط به Serialize و Deserialize سندهای XML را در خود جای داده است. نمونه ی زیر روش Serialize کردن یک آبجکت را نشان می دهد:

```
XmlSerializer xmSer = new XmlSerializer(typeof(Person));
FileStream st = new FileStream("C:\\Test.xml", FileMode.OpenOrCreate);
Person p = new Person("Ozhan Ostovar", 44);
xmSer.Serialize(st, p);
st.Close();
```

برای Deserialize کردن آن به روش زیر عمل می کنیم:

```
XmlSerializer xmSer = new XmlSerializer(typeof(Person));
FileStream st = new FileStream("C:\\Test.xml", FileMode.Open);
Person p = xmSer.Deserialize(st) as Person;
st.Close();
```

فرم XmlSerializerSample.cs مثالی دیگر از کلاس فوق را نمایش می دهد که مطالعه آن به درک بهتر مطلب کمک خواهد کرد.

XPathDocument و XPathNavigator

همان طور که در توضیحات XPATH گفته شد توسط این ویژگی می توان به قسمتی از یک سند XML اشاره کرد. به عبارتی دیگر XPATH زبانی برای استخراج از ساختار XML است و کلاس های XPathDocument و XPathNavigator برای این منظور ایجاد شده اند. این دو کلاس در فضای System.Xml.XPath قرار دارند.

فرم XPathSample.cs نمونه ای از کاربرد کلاس های فوق را نشان می دهد. در این مثال ابتدا بعد از اضافه کردن namespace های مورد نیاز به تعریف نمونه ای از XPathDocument می پردازیم و سند XML مورد نظر را به آن نسبت می دهیم. این کلاس از تکنولوژی XSLT جهت بارگذاری سریع سند و انجام بهینه ی پردازش ها استفاده می کند. از کلاس XPathNavigator نمونه ای را می سازیم. این کلاس به ما قدرت حرکت بین Node ها و Attribute ها را می دهد. متد هایی مانند MoveToRoot ، MoveToNext و... امکان حرکت را فراهم می آورند.

```
XPathDocument docNav = new XPathDocument(Application.StartupPath +
"\\XMLWriter.xml");
XPathNavigator nav = docNav.CreateNavigator();
nav.MoveToRoot();

//Move to the first child node (comment field).
```

```
nav.MoveToFirstChild();

do
{
    //Find the first element.
    if (nav.NodeType == XPathNodeType.Element)
    {
        //Determine whether children exist.
        if (nav.HasChildren == true)
        {

            //Move to the first child.
            nav.MoveToFirstChild();

            //Loop through all of the children.
            do
            {
                //Display the data.
                MessageBox.Show("The XML string for this child ");
                MessageBox.Show("is '" + nav.Value + "'");

                //Check for attributes.
                if (nav.HasAttributes == true)
                {
                    MessageBox.Show("This node has attributes");
                }
            } while (nav.MoveToNext());
        }
    }
} while (nav.MoveToNext());
```

بخش هفتم

چند مثال کاربردی

در این بخش قصد داریم درباره برخی از کلاس ها و مطالب گفته شده در فصل قبل مثال هایی را ارائه دهیم تا مفاهیم هر چه بهتر و بیشتر روشن شوند. در هر مثال توضیحات لازم ارائه خواهد شد. و اکثر مثال ها در نرم افزار ارائه شده پیاده سازی شده اند.

خوراک RSS

ویکی پدیا Rss را اینگونه شرح می دهد " خانواده‌ای از قالب‌های خورد فید در وب است که برای انتشار محتویاتی که در بازه‌های زمانی خاص به‌روزرسانی می‌شوند استفاده می‌شود (مانند وب‌نوشت‌ها، عناوین اخبار، و پادکست‌ها) " Rss مخفف Rich Site Summary به معنی " چکیده سایت " یا " مختصر و مفید سایت " است و به شما امکان می‌دهد تا از آخرین خبرها و عناوین سایت و یا وبلاگهای مورد علاقه خود با خبر باشید، بدون آنکه نیاز داشته باشید به آنها سر بزنید. Rss چیزی نیست به جز یک نشانه‌گذاری استاندارد شده XML، به وسیله آن می توانید محتوایی را که قصد به اشتراک گذاشتن آن را دارید توصیف کنید. به این ترتیب با قرار دادن یک قسمت Rss سایت در متن سایت دیگر، با عوض شدن اخبار و عناوین سایت اول، این محتویات در سایت دوم به صورت خودکار عوض می‌شوند.

خبر خوان (Rss Reader)

یک نرم افزار RSS خوان الکترونیکی است. اکنون نرم افزارهای بسیاری در این زمینه وجود دارند که می‌توان با دانلود آنها RSS خواند. مرورگرهای اینترنتی نیز به سیستم RSS خوانی مجهز شده‌اند.

فرم RSSReader.cs نمونه ای از یک نرم افزار خبر خوان می باشد که با بررسی آن می توانید نحوه ساخت یک خبرخوان را بیاموزید. همان طوری که ملاحظه می کنید از کلاس XPathNavigator جهت حرکت بین نودهای سند Rss که (در فرمت XML است) استفاده شده. هر خط از کد ها توسط کامنت هایی توضیح داده شده است.

```
try
{
    // Clear the treeview.
    tvwRss.Nodes.Clear();
    // set the wait cursor
    this.Cursor = Cursors.WaitCursor;
    // create a new xml doc
    XmlDocument doc = new XmlDocument();
    try
    {
        // load the xml doc
```

```
            doc.Load(txtRssAddress.Text);
            // return the cursor
            this.Cursor = Cursors.Default;
        }
        catch (Exception ex1)
        {
            // return the cursor
            this.Cursor = Cursors.Default;
            // tell a story
            MessageBox.Show(ex1.Message);
            return;
        }

        // get an xpath navigator
        XPathNavigator navigator = doc.CreateNavigator();
        try
        {
            // look for the path to the rss item titles; navigate
            // through the nodes to get all titles
            XPathNodeIterator nodes =
            navigator.Select("//rss/channel/item/title");
            while (nodes.MoveNext())
            {
                // clean up the text for display
                XPathNavigator node = nodes.Current;
                string tmp = node.Value.Trim();
                tmp = tmp.Replace("\n", "");
                tmp = tmp.Replace("\r", "");

                // add a new treeview node for this
                // news item title
                tvwRss.Nodes.Add(tmp);
            }

            // set a position counter
            int position = 0;
            // Get the links from the RSS feed
            XPathNodeIterator nodesLink =
navigator.Select("//rss/channel/item/link");
            while (nodesLink.MoveNext())
            {
                // clean up the link
                XPathNavigator node = nodesLink.Current;
                string tmp = node.Value.Trim();
                tmp = tmp.Replace("\n", "");
                tmp = tmp.Replace("\r", "");
                // use the position counter
                // to add a link child node
                // to each news item title
                tvwRss.Nodes[position].Nodes.Add(tmp);
                // increment the position counter
                position++;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "RSS Feed Load Error");
        }
        // restore the cursor
        this.Cursor = Cursors.Default;
        tvwRss.ExpandAll();
```

```
}
catch (Exception ex2)
{
    // snitch
    MessageBox.Show(ex2.ToString(), "RSS Feed Initialization Failure");
}
```

نمایش و ویرایش یک سند XML در DataGridView

در ادامه قصد داریم به بررسی فرم ReadXMLDataGridView.cs بپردازیم، سند XML را به کمک کنترل DataGridView نمایش دهیم و مقادیر آن را حذف ، ویرایش و سطر جدیدی به آن اضافه نماییم.

ابتدا پس از انتخاب فایل توسط کاربر با تعریف یک Dataset با نام ds و استفاده از متد ReadXml فایل مورد نظر را خوانده و مقدار پراپرتی DataSource گریدویو را برابر آن (ds.Tables[0]) قرار می دهیم. مقدار primary key را برای جدول مشخص می نماییم تا در جستجو و حذف رکورد ها از آن استفاده شود. در اینجا مقدار Email به عنوان pk در نظر گرفته شده است.

```
private void btnLoad_Click(object sender, EventArgs e)
{
    openFileDialog1.FileName = "ostovarit.xml";
    openFileDialog1.Filter = "XML Files(ostovarit.xml)|ostovarit.xml";
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        Path = openFileDialog1.FileName;
        ds = new DataSet();
        ds.ReadXml(Path);
        dataGridView1.DataSource = ds.Tables[0];
        ds.Tables[0].Constraints.Add("E-mail", ds.Tables[0].Columns[1],
true);
        fdir = Path.Substring(0, Path.LastIndexOf("\\") + 1);
        if (dataGridView1.RowCount > 0)
            this.dataGridView1.ClearSelection();
    }
}
```

در ایونت dataGridView1_CellClick متد showdata را صدا می زنیم . این متد اطلاعات هر ردیف از گرید ویو را بعد از کلیک کاربر روی سطر ها در تکست باکس های مربوطه نمایش می دهد. اما قبل از انجام این عملیات بررسی می شود که فایل XML بارگذاری شده حاوی اطلاعات باشد.

```
void showdata()
{
    if (ds.Tables[0].Rows.Count > 0)
    {
        rno = dataGridView1.CurrentRow.Index;
        pictureBox1.Image = null;
        txtFullName.Text = ds.Tables[0].Rows[rno][0].ToString();
        txtEmail.Text = ds.Tables[0].Rows[rno][1].ToString();
        txtMob.Text = ds.Tables[0].Rows[rno][2].ToString();
        pictureBox1.ImageLocation = fdir + dataGridView1[3,
        dataGridView1.CurrentRow.Index].Value.ToString();
    }
```

```
    else
    {
        MessageBox.Show("No records. Create another XML file.");
        File.Delete(Application.StartupPath + "\\ostovarit.xml");
        ClearForm();
    }
}
```

برای ایجاد رکورد جدید ابتدا با استفاده از متد Find پست الکترونیک وارد شده در تکست باکس را در Dataset جستجو می کنیم و حاصل جستجو را در یک Datarow نگه میداریم. قبل از عملیات Insert با چک کردن Datarow (در صورت خالی بودن آن) از تکراری نبودن پست الکترونیک اطمینان حاصل می کنیم. سپس با متد NewRow سطر جدیدی از Dataset ساخته و مقادیر تکست باکس ها را به سطر جدید اضافه میکنیم. در نهایت با استفاده از متد Add سطر جدید را در Dataset ذخیره میکنیم. تا این مرحله اطلاعات در Dataset ذخیره شده است، برای ذخیره سطر جدید روی فایل XML از متد WriteXml استفاده می شود. فعالیت های صورت گرفته با کمی تفاوت در زمان ویرایش اطلاعات نیز تکرار می شود. در زمان ویرایش دیگر نیاز به ایجاد سطر جدید نخواهد بود و تغییرات مستقیما روی Dataset صورت می گیرند.

```
private void btnInsert_Click(object sender, EventArgs e)
{
    dr = null;
    dr = ds.Tables[0].Rows.Find(txtEmail.Text.Trim());
    if (dr == null)
    {
        dr = ds.Tables[0].NewRow();
        dr[0] = txtFullName.Text.Trim();
        dr[1] = txtEmail.Text.Trim();
        dr[2] = txtMob.Text.Trim();
        dr[3] = txtEmail.Text.Trim() + ".jpg";
        phototask();
        ds.Tables[0].Rows.Add(dr);
        dataGridView1.DataSource = ds.Tables[0];
        ds.WriteXml(Path);
    }
    else
        MessageBox.Show("Insert another Email address", "Error",
        MessageBoxButtons.OK,MessageBoxIcon.Error);
}

void phototask()
{
    //finding name of photo ,saving it in directory of xml file with unique
    name(Email+photo name)
    string pdestin = Application.StartupPath + "\\" + txtEmail.Text.Trim()
    + ".jpg ";
    pictureBox1.Image.Save(pdestin);//saving photo on disk
}

private void btnUpdate_Click(object sender, EventArgs e)
{
    DataRow drow =
    ds.Tables[0].Rows.Find(dataGridView1[1,dataGridView1.CurrentRow.Index].
    Value.ToString());
    if (drow != null)
    {
        rno = ds.Tables[0].Rows.IndexOf(drow);
```

```
        ds.Tables[0].Rows[rno][0] = txtFullName.Text;
        ds.Tables[0].Rows[rno][1] = txtEmail.Text;
        ds.Tables[0].Rows[rno][2] = txtMob.Text;
        phototask();
        ds.Tables[0].Rows[rno][3] = txtEmail.Text.Trim() + ".jpg";
        ds.WriteXml(Path);
        MessageBox.Show("record updated");
    }
    else
        MessageBox.Show("no record exists with this Email.");
}
```

در زمان حذف با متد Find کلید جدول را جستجو کرده و سطر مورد نظر را در یک Datarow نگاه می داریم. سپس با استفاده از متد Remove سطر را از Dataset حذف و در نهایت تغییرات را با متد WriteXml روی سند XML اعمال می کنیم.

```
if (dataGridView1.SelectedRows.Count > 0)
{
    DataRow drow = ds.Tables[0].Rows.Find(dataGridView1[1,
    dataGridView1.CurrentRow.Index].Value.ToString());
    if (drow != null)
    {
        File.Delete(Application.StartupPath + "\\" + dataGridView1[3,
        dataGridView1.CurrentRow.Index].Value.ToString());
        ds.Tables[0].Rows.Remove(drow);
        ds.WriteXml(Path);
        MessageBox.Show("record deleted.");
        showdata();
    }
    else
        MessageBox.Show("no record exists with this Email.");
}
```

نمایش یک سند XML در WebBrowser

فرم ReadXMLWebBrowser.cs یک مثال از نحوه نمایش فایل XML در کنترل webBrowser می باشد.

```
openFileDialog1.FileName = ".xml";
openFileDialog1.Filter = "XML Files(*.xml)|*.xml";
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    txtRoot.Text = openFileDialog1.FileName;
    StreamReader sr = new StreamReader(txtRoot.Text);
    webBrowser1.DocumentText= sr.ReadToEnd();
}
```

همان طور که ملاحظه می کنید، کاربر پس از انتخاب سند XML توسط یک StreamReader محتویات را خوانده و پراپرتی DocumentText مرورگر را برابر آن قرار می دهد.

تلفیق دو سند XML

فرم MergeXML.cs روش تلفیق (یکی کردن) دو سند XML و ذخیره آنها را نمایش می دهد. ابتدا هر دو فایل را با استفاده از متد ReadXml در یک Dataset بارگذاری نموده سپس متد WriteXml را برای ایجاد فایل نهایی می نویسیم.

```
DataSet ds = new DataSet();
ds.ReadXml(textBox1.Text);
ds.ReadXml(textBox2.Text);
ds.WriteXml(Application.StartupPath + @"\Merge.xml");
```

خروجی از پایگاه داده به فرمت XML

یکی از مواردی که در نرم افزار ها مورد استفاده قرار میگیرد امکان خروجی گرفتن یا پشتیبانی از جدول های پایگاه داده به فرمت XML می باشد. برای این منظور کامپوننت های آماده ای وجود دارد که امکانات بی شماری را در اختیار برنامه نویسان قرار می دهد، کد زیر این مطلب را به طور ساده به نمایش گذاشته است:

```
SqlConnection con = new SqlConnection(@"Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\TestDB.mdf;Integrated
Security=True;User Instance=True");
SqlDataAdapter da = new SqlDataAdapter("Select * from Customers", con);
DataSet ds = new DataSet();
da.Fill(ds, "Customers");
ds.WriteXml(Application.StartupPath +"//ConvertSql2Xml.xml",
XmlWriteMode.IgnoreSchema);
```

برای این منظور از متد WriteXml مربوط به DataSet استفاده کرده و از جدول Customers که در پایگاه داده TestDB قرار دارد به فرمت XML خروجی گرفتیم. فرم ExportSQL2XML.cs نمونه ای از کد فوق را نمایش می دهد.

Virtual Token Descriptor (VTD)

کتابخانه ای برای کار با سند های XML می باشد. این کتابخانه متن باز بوده و از طریق آدرس زیر قابل دسترسی است.
http://vtd-xml.sourceforge.net

کتابخانه VTD در پوشه dll قرار دارد. فرم VTDReadOnly.cs نمونه ای از روش استفاده این کتابخانه را نشان می دهد. این کتابخانه قابل استفاده در زبان های C# ، C و Java می باشد.

امیدوارم با مطالعه مطالب ذکر شده در این هفت بخش و بررسی مثال های نرم افزار تسلط لازم برای کار با اسناد XML را بدست آورده باشید. برای کسب اطلاعات بیشتر لینک های ارائه شده در نرم افزار را مطالعه نمایید. با عضویت در وب سایت www.ostovarit.com از ویرایش های جدید این کتاب و انتشار مقالات و کتاب های دیگر توسط فناوری اطلاعات استوار مطلع شوید.

نظرات و پیشنهادات خود را به پست الکترونیک ostovarit@gmail.com ارسال نمایید.