ASP.NET XML Kullanımı

XML dosyaları, ASP.NET ve diğer dillerde farklı amaçlar için sıklıkla kullanılmaktadır. Bir çok işlem XML dosyaları üzerinden yapılır. Örneğin **web.config** dosyası XML formatındadır. **DataSet** sınıfı herhangi bir data'yı XML formatında temsil edebilmektedir. Bunun bazı faydaları şunlardır:

1. Database'den alınan data'yı XML formatına dönüştürüp başka programların kullanması sağlanabilir.
2. **AdRotator** control'ünün kullanacağı bir dosya tipidir.
3. ASP.NET uygulamasında **site map** yaparkenæ, kullanılmaktadır.
4. **Serialization** işlemlerinde görev almaktadır. XML standart bir yol sunduğu için compatibility(uyumluluk)æ, hatalarından kurtulmuş oluruz.

XML case sensitive'dir. Yani <ID> ile <id> farklı elementlerdir.

XML dosyaları **genelde** aşağıdaki satır ile **başlar:**

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

Not: Bu satırın XML dosyalarında olması zorunlu değildir.

Attributes

Bir XML elementininin extra bilgilerini tutar.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--Xml Calismalar Dosyasi-->
<Asp.NetEgitim xmlns="http://www.w3.org/XML/1998/namespaceim">
<Sayfa ID="1" Format=".aspx"> <!--ID ve Format birer attribute'tir.-->
   <Icerik>Div Kullanimi</Icerik>
   <Icerik>Panel Kullanimi</Icerik>
</Sayfa>
<Sayfa ID="2" Format=".htm">
   <Icerik>HTML Div Kullanimi</Icerik>
   <Icerik>HTML Panel Kullanimi</Icerik>
   <Icerik>HTML Submit Buttonu</Icerik>
</Sayfa>
</Asp.NetEgitim>
```

Attribute'lerin sıralaması önemli değildir. Yani ß,IDß ile ß,Formatß yer değiştirebilir.

Yorum Yapmak İçin Kullanılan Tag

```
<!--Xml Calismalar Dosyasi-->
```

System.Xml Namespace

System.Xml namespace'i içerisinde XML işlemlerini gerçekleştiren sınıflar vardır. Bu sınıflar ile aşağıdaki işlemler yapılır:

1. Read
2. Write
3. Manipulate
4. Validate
5. Showing Information

XmlTextWriter

TEXT türündeki dosyalarda okuma, yazma işlemlerini **StreamWriter**, **StreamReader** sınıfları sağlarken, XML dosya okuma, yazma işlemlerini **XmlTextWriter** ve **XmlTextReader** sınıfları sağlar.

Örnek:

```
protected void Page_Load(object sender, EventArgs e)
{
   string path = Path.Combine(Request.PhysicalApplicationPath, @"App_Data\XMLCalismalar.xml");

   FileStream stream = new FileStream(path, FileMode.Create);
```

```
    XmlTextWriter xmlWrite = new XmlTextWriter(stream, null);

    xmlWrite.WriteStartDocument();

    xmlWrite.WriteStartElement("SuperProProductList");
    xmlWrite.WriteComment("This file is generated by the XmlTextWriter class");
    //write first product
    xmlWrite.WriteStartElement("Product");
    xmlWrite.WriteAttributeString("ID", "1");
    xmlWrite.WriteAttributeString("Name", "Chair");
    xmlWrite.WriteStartElement("Price");
    xmlWrite.WriteString("49.33");
    xmlWrite.WriteEndElement();//Price elementinin sonu
    xmlWrite.WriteEndElement();//Product elementinin sonu
    xmlWrite.WriteEndElement();//SuperProProductList elementinin sonu
    xmlWrite.WriteEndDocument();//Dosyanın sonu
    xmlWrite.Close();
    stream.Close();
}
```

Oluşan XML dosyasının içeriği aşağıdaki gibi olur:

```
<?xml version="1.0"?>
<SuperProProductList>
<!--This file is generated by the XmlTextWriter class-->
<Product ID="1" Name="Chair">
   <Price>49.lt;/Price>
</Product>
</SuperProProductList>
```

XmlTextReader

Bu sınıfın bazı önemli metod ve property'leri şunlardır:

1. Read() : Bir sonraki node başarılı bir şekilde okunmuşsa **true** dönderir, aksi taktirde **false** dönderir.

2. Name : Node'nin ismini dönderir.

3. Value : `<x>Bu bir içeriktir</x>` örneğindeki gibi **open** ve **close** tag'leri arasındaki değeri dönderir. Örneğimizdeki bu değer: `"Bu bir içeriktir"`

4. AttributeCount : Bir elementin attribute **sayısını** dönderir.

5. GetAttribute() : Parametre olarak **index** numarası veya **string** alır. Örneğin, `GetAttribute(0)` denildiğinde sıfırıncı index'teki attribute'i dönderir.

6. Skip() : O anki node'nin çocuklarını skip eder.

7. ResetState() : Başa sarar.

8. NodeType : **XmlNodeType enum** değerlerinden birini alır. **attribute, comment, element** gibi tiplere sahip olduğu için önemlidir. Bir **if** döngüsü içerisinde **NodeType** gerekli olabilmektedir.

Working with XML Documents in Memory

XmlDocument sınıfı **XmlTextWriter** sınıfına benzer; fakat bu sınıf **gelişmiş editing** özelliklerine sahiptir. **XmlDocument** sınıfından yaratılan bir nesne kullanarak, XML dosyasının herhangi bir yerini **değiştirebilir, silebilir, ekleme** yapabiliriz.

XmlTextWriter sadece **yazmayı**, **XmlTextReader** ise sadece **okumayı** sağlarken **XmlDocument yazmayı**, **okumayı** ve **değiştirmeyi** sağlar. Ayrıca **XmlDocument**, **XmlTextWriter** gibi yeni bir XML dosyası yaratmayı da sağlar.

Örnek:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Start with a blank in-memory document.
    XmlDocument doc = new XmlDocument();
    // Create some variables that will be useful for
    // manipulating XML data.
    XmlElement rootElement, productElement, priceElement;
    XmlAttribute productAttribute;
```

```
    XmlComment comment;
    // Create the declaration.
    XmlDeclaration declaration;
    declaration = doc.CreateXmlDeclaration("1.0", null, "yes");
    // Insert the declaration as the first node.
    doc.InsertBefore(declaration, doc.DocumentElement);
    // Add a comment.
    comment = doc.CreateComment("Created with the XmlDocument class.");
    doc.InsertAfter(comment, declaration);
    // Add the root node.
    rootElement = doc.CreateElement("SuperProProductList");
    doc.InsertAfter(rootElement, comment);
    // Add the first product.
    productElement = doc.CreateElement("Product");
    rootElement.AppendChild(productElement);
    // Set and add the product attributes.
    productAttribute = doc.CreateAttribute("ID");
    productAttribute.Value = "1";
    productElement.SetAttributeNode(productAttribute);
    productAttribute = doc.CreateAttribute("Name");
    productAttribute.Value = "Chair";
    productElement.SetAttributeNode(productAttribute);
    // Add the price node.
    priceElement = doc.CreateElement("Price");
    priceElement.InnerText = "49.33";
    productElement.AppendChild(priceElement);
    // (Code to add two more products omitted.)
    // Save the document.
    string file = Path.Combine(Request.PhysicalApplicationPath,
    @"App_Data\SuperProProductList.xml");
    doc.Save(file);
}
```

Searching

XmlDocument sınıfının **searching** desteği de bulunmaktadır. Bunun için, **GetElementByID**() ve **GetElementByTagName**() metodları kullanılır. Sonuçları bir **listede** tutmak için **XmlNodeList** nesnesi kullanılır.

Örnek:

```
protected void Page_Load(object sender, EventArgs e)
{
    string path = Path.Combine(Request.PhysicalApplicationPath, @"App_Data\XMLCalismalar.xml");

    FileStream stream = new FileStream(path, FileMode.Open);


    //A node includes comments, whitespace, opening tags, closing tags,
    //content, and even the XML declaration at the top of your file.

    XmlDocument doc = new XmlDocument();
    doc.Load(stream);

    XmlNodeList aramaSonucuNode = doc.GetElementsByTagName("Sayfa");

    foreach (XmlNode item in aramaSonucuNode)
    {
        lblXml.Text += item.FirstChild.Value + "<br />";
    }
    stream.Close();
}
```

Not: FirstChild property ile alakalı şunları bilmekte fayda vardır: Eğer **XmlNode** <sayfa> olursa **item.FirstChild** <içerik> dönderir. Bunun değeri **null** olur. Fakat <içerik> **XmlNode** ise **item.FirstChild** , **"ASP"** ifadesini dönderir. Bunun değeri ise **Asp** dir.

XML Validation

XML dosyasını kendimiz için oluşturduğumuz zaman, **XSD** dosyası yaratmamıza gerek yoktur; çünkü XML dosyasında yazdığımız elementlere, attribute'lere vs göre uygun kodlar yazabiliriz. Fakat bir başkası XML dosyasını kullanmak istediğinde problemsiz kullanabilecek mi? XML dosyası bir kurala göre hazırlanmamışsa, XML dosyasını çalıştıracak uygun kodlar da **yazılamaz**. Bunun için, **.xsd** uzantılı bir dosya oluşturularak, XML dosyasının **sentaks** yapısı belirtilir.

Xml Namespace: .xsd uzantılı dosyalar oluşturabilmek için önce XML namespace'in ne olduğunu bilmek gerekir. Nasıl ki bir sınıf yarattığımızda o sınıf bir **namespace** içerisinde tutuluyorsa, yarattığımız XML elementleri de bir **namespace** içerisinde bulunur.
Namespace ismi genelde **sitemizin** ismi ile aynıdır. Bir XML dosyasında her bir element bir namespace'e ait olur. Yani, XML dosyasındaki elementlere farklı namespace'ler verebiliriz.

Örnek:

XSD Dosyası:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="XMLCalismalarSchema"
   targetNamespace="http://www.w3.org/XML/1998/namespace"
   elementFormDefault="qualified"
   xmlns="http://tempuri.org/XMLCalismalarSchema.xsd"
   xmlns:mstns="http://tempuri.org/XMLCalismalarSchema.xsd"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
<xs:element name="Asp.NetEgitim"><!--Root elementi yazmak zorundayiz.-->
   <xs:complexType><!--Bir element baska elementlere sahipse complexType olur.-->
    <!--sequence ile root element'inin sayısız elementlerinin olacagini ifade ederi.-->
    <!--sequence baslangic bitis tagleri arasina alt elementler yazilir.-->
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="Sayfa">
       <xs:complexType>
         <xs:sequence>
         <xs:element name="Icerik" type="xs:string"/>
         </xs:sequence>
         <xs:attribute name="ID" use="required" type="xs:string" />
         <xs:attribute name="Format" use="required" type="xs:string" />
       </xs:complexType>
      </xs:element>
    </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:schema>
<!--bir element tanimlanirken :complexType denmisse bu element içinde mutlaka element tanimlamaliyiz.-->
```

XML Dosyası:

```
<!---->
<?xml version="1.0" encoding="utf-8" ?>
<!--Xml Calismalar Dosyasi-->
<Asp.NetEgitim xmlns="http://www.w3.org/XML/1998/namespaceim">
<Sayfa ID="1" Format=".aspx">
   <Icerik>Div Kullanimi</Icerik>
   <Icerik>Panel Kullanimi</Icerik>
</Sayfa>
</Asp.NetEgitim>
```

Validating an Xml Document

XSD dosyası ile bir XML dosyasının sentakslarını doğru şekilde konfigüre edip etmediğimizi anlayabilmek için şu tarz bir kod yazmak gerekir:

```
protected void Page_Load(object sender, EventArgs e)
{
   string filePath = Request.PhysicalApplicationPath + @"\App_Data\XMLCalismalar.xml";


   // Set the validation settings.
   XmlReaderSettings settings = new XmlReaderSettings();
   settings.Schemas.Add("http://www.w3.org/XML/1998/namespace",
   Request.PhysicalApplicationPath + @"\App_Data\XMLCalismalarSchema.xsd");
   settings.ValidationType = ValidationType.Schema;


   // Open the XML file.
   FileStream fs = new FileStream(filePath, FileMode.Open);

   // Create the validating reader.
   XmlReader r = XmlReader.Create(fs, settings);
```

```
    // Read through the document.
    while (r.Read())
    {
     // Process document here.
     // If an error is found, an exception will be thrown.
    }
    fs.Close();
}
```