Practical 3
XML and Associated Schema
JSON
WEB API Input Validation (Using Framework/Pipeline)

 **Objectives**

Creating an XML File with an Associated XML Schema

You will learn to:
- Create XML file using XML editor
- Create XML file using XmlTextWriter class
- Create XML file and XML schema
- Associate an XML file and schema
- Convert XML to JSON…
- Using model annotation for Web API validation

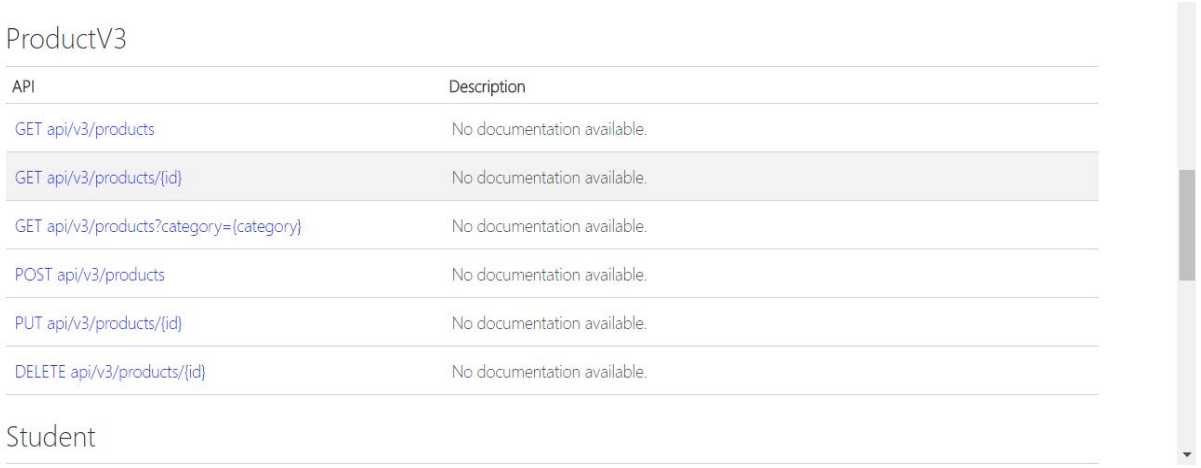
This walkthrough will illustrate the capabilities of working with XML files that have associated schemas. It will take you through the process of creating an XML file and an XML Schema based on the XML Schema definition language (XSD). Then you will create an association between the XML Schema and the XML file. After this association is established, you will work on the XML file in XML view in order to see the advanced capabilities of the XML editor. For this walkthrough the XML file will represent a list of companies. You will see how the XML file contains the data whereas the XML Schema only defines that data.

Note: By associating an XML Schema with your XML file, statement completion is enabled in the XML editor.

PART 1

Tasks

When a client sends data to your web API, often you want to validate the data before doing any processing. This practical shows how to annotate your models, use the annotations for data validation, and handle validation errors in your web API.



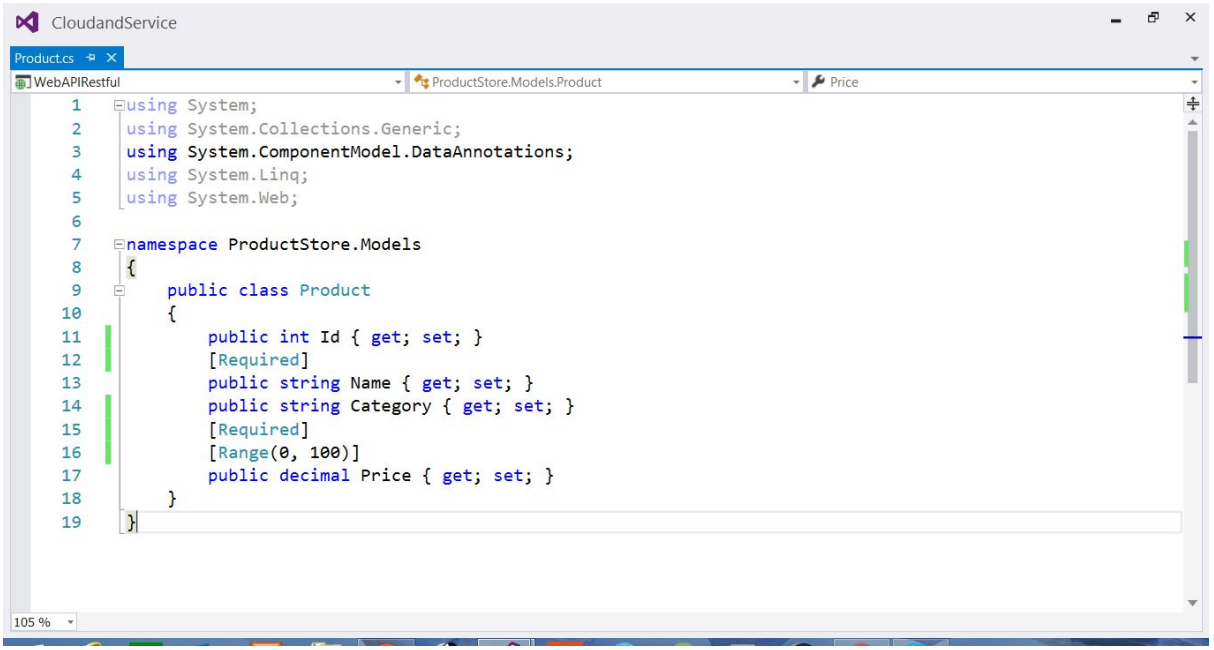
ProductV3

API	Description
GET api/v3/products	No documentation available.
GET api/v3/products/{id}	No documentation available.
GET api/v3/products?category={category}	No documentation available.
POST api/v3/products	No documentation available.
PUT api/v3/products/{id}	No documentation available.
DELETE api/v3/products/{id}	No documentation available.

Student

1. Data Annotations

In ASP.NET Web API, you can use attributes from the System.ComponentModel.DataAnnotations namespace to set validation rules for properties on your model. Consider the following model:



Add an interface ***IProductRepository*** and ***ProductRepository*** class in the models folder:

```
namespace ProductStore.Models
{
    interface IProductRepository
    {
        IEnumerable<Product>
GetAll();
        Product Get(int id);
        Product Add(Product item);
        void Remove(int id);
        bool Update(Product item);
    }
}
```

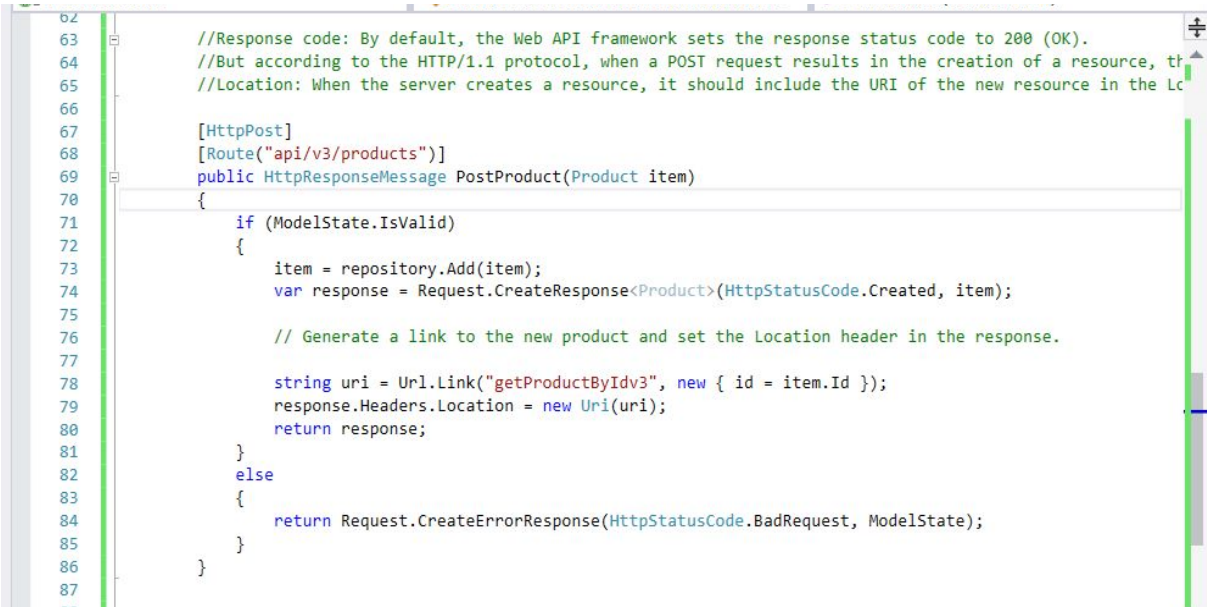
```
namespace ProductStore.Models
{
    public class ProductRepository :
IProductRepository
    {
        private List<Product> products = new
List<Product>();
        private int _nextId = 1;

        public ProductRepository()
        {
            Add(new Product { Name = "Tomato
soup", Category = "Groceries", Price = 1.39M });
            Add(new Product { Name = "Yo-yo",
Category = "Toys", Price = 3.75M });
            Add(new Product { Name = "Hammer",
Category = "Hardware", Price = 16.99M });
        }

        public IEnumerable<Product> GetAll()
        {
            return products;
        }

        public Product Get(int id)
        {
            return products.Find(p => p.Id ==
id);
        }

        public Product Add(Product item)
        {
            if (item == null)
            {
                throw new
```

```
ArgumentNullException("item");
            }

            item.Id = _nextId++;
            products.Add(item);
            return item;
        }

        public void Remove(int id)
        {
            products.RemoveAll(p => p.Id == id);
        }

        public bool Update(Product item)
        {
            if (item == null)
            {
                throw new
ArgumentNullException("item");
            }
            int index = products.FindIndex(p =>
p.Id == item.Id);
            if (index == -1)
            {
                return false;
            }
            products.RemoveAt(index);
            products.Add(item);
            return true;
        }
    }

}
```

2. Handling Validation Errors

Web API does not automatically return an error to the client when validation fails. It is up to the controller action to check the model state and respond appropriately.

Create a new Web API Controller, ProductV3Controller



```
62
63          //Response code: By default, the Web API framework sets the response status code to 200 (OK).
64          //But according to the HTTP/1.1 protocol, when a POST request results in the creation of a resource, th
65          //Location: When the server creates a resource, it should include the URI of the new resource in the Lc
66
67          [HttpPost]
68          [Route("api/v3/products")]
69          public HttpResponseMessage PostProduct(Product item)
70          {
71              if (ModelState.IsValid)
72              {
73                  item = repository.Add(item);
74                  var response = Request.CreateResponse<Product>(HttpStatusCode.Created, item);
75
76                  // Generate a link to the new product and set the Location header in the response.
77
78                  string uri = Url.Link("getProductByIdv3", new { id = item.Id });
79                  response.Headers.Location = new Uri(uri);
80                  return response;
81              }
82              else
83              {
84                  return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ModelState);
85              }
86          }
87
```

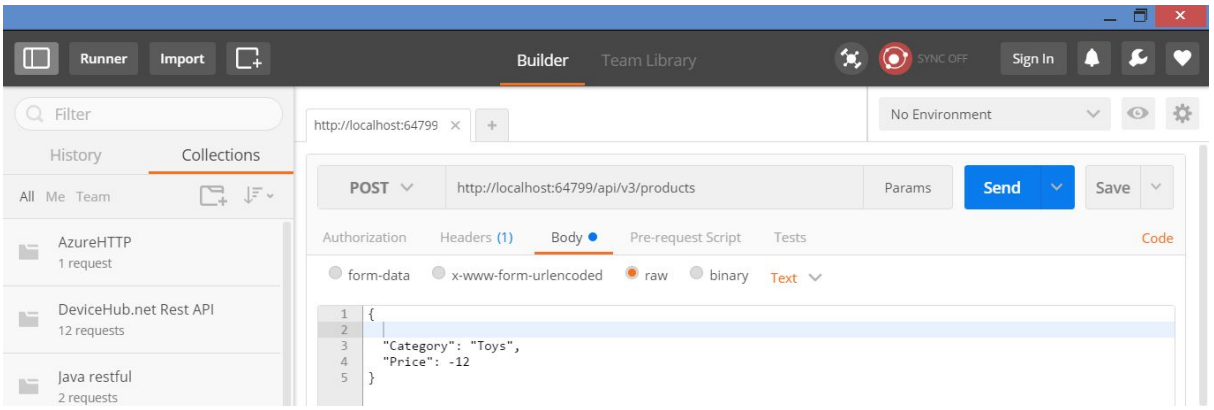
```
 public class ProductV3Controller : ApiController
    {
        static readonly IProductRepository repository = new ProductRepository();


        //Version 3
        //[Authorize]
        [HttpGet]
        [Route("api/v3/products")]
        public IEnumerable<Product> GetAllProductsFromRepository()
        {
            return repository.GetAll();

        }
        //Route constraints let you restrict how the parameters in the route template
are matched.
        //The general syntax is "{parameter:constraint}".
        //Constraints on URL parameters

        //We can even restrict the template placeholder to the type of parameter it can
have.
        //For example, we can restrict that the request will be only served if the id is
greater than 2.
        //Otherwise the request will not work. For this, we will apply multiple
constraints in the same request:


        //Type of the parameter id must be an integer.
        //id should be greater than 2.
        //http://localhost:9000/api/v3/products/1 404 error code
        [HttpGet]
        [Route("api/v3/products/{id:int:min(2)}", Name = "getProductByIdv3")]

        public Product retrieveProductfromRepository(int id)
        {
            Product item = repository.Get(id);
            if (item == null)
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
            return item;
        }


        [HttpGet]
        [Route("api/v3/products", Name = "getProductByCategoryv3")]
        //http://localhost:9000/api/v3/products?category=
        //http://localhost:9000/api/v3/products?category=Groceries

        public IEnumerable<Product> GetProductsByCategory(string category)
        {
            return repository.GetAll().Where(
                p => string.Equals(p.Category, category,
StringComparison.OrdinalIgnoreCase));
        }


        //Response code: By default, the Web API framework sets the response status code
to 200 (OK).
        //But according to the HTTP/1.1 protocol, when a POST request results in the
```

```
creation of a resource, the server should reply with status 201 (Created).
        //Location: When the server creates a resource, it should include the URI of the
new resource in the Location header of the response.

        [HttpPost]
        [Route("api/v3/products")]
        public HttpResponseMessage PostProduct(Product item)
        {
            if (ModelState.IsValid)
            {
                item = repository.Add(item);
                var response = Request.CreateResponse<Product>(HttpStatusCode.Created,
item);

                // Generate a link to the new product and set the Location header in the
response.

                string uri = Url.Link("getProductByIdv3", new { id = item.Id });
                response.Headers.Location = new Uri(uri);
                return response;
            }
            else
            {
                return Request.CreateErrorResponse(HttpStatusCode.BadRequest,
ModelState);
            }
        }









        [HttpPut]
        [Route("api/v3/products/{id:int}")]
        public void PutProduct(int id, Product product)
        {
            product.Id = id;
            if (!repository.Update(product))
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
        }

        [HttpDelete]
        [Route("api/v3/products/{id:int}")]
        public void DeleteProduct(int id)
        {
            repository.Remove(id);
        }
    }
```

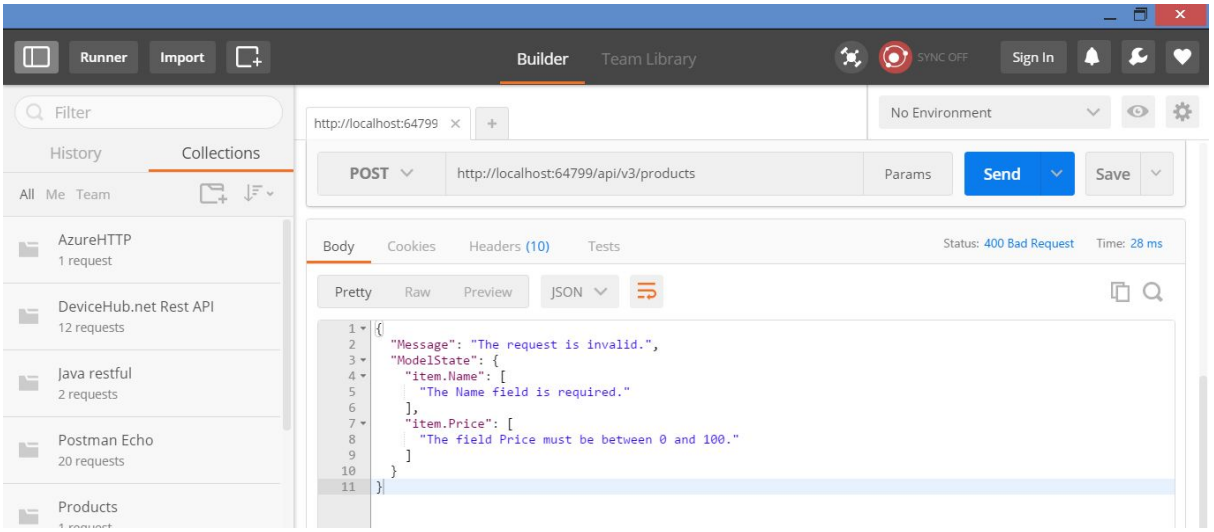
3. Testing the Web API

"Under-Posting": Under-posting happens when the client leaves out some properties

Send a request without name:

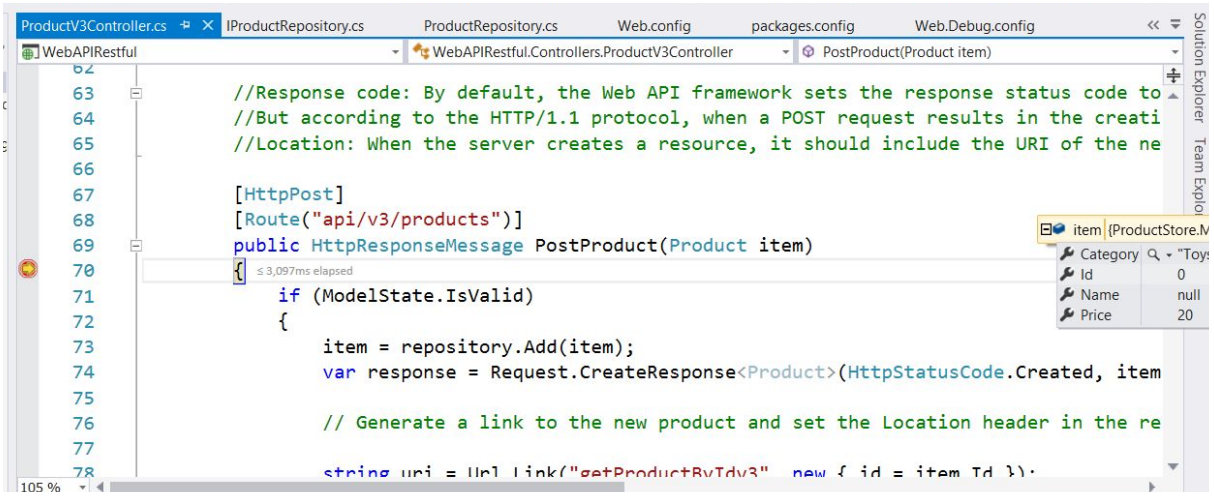


Response:



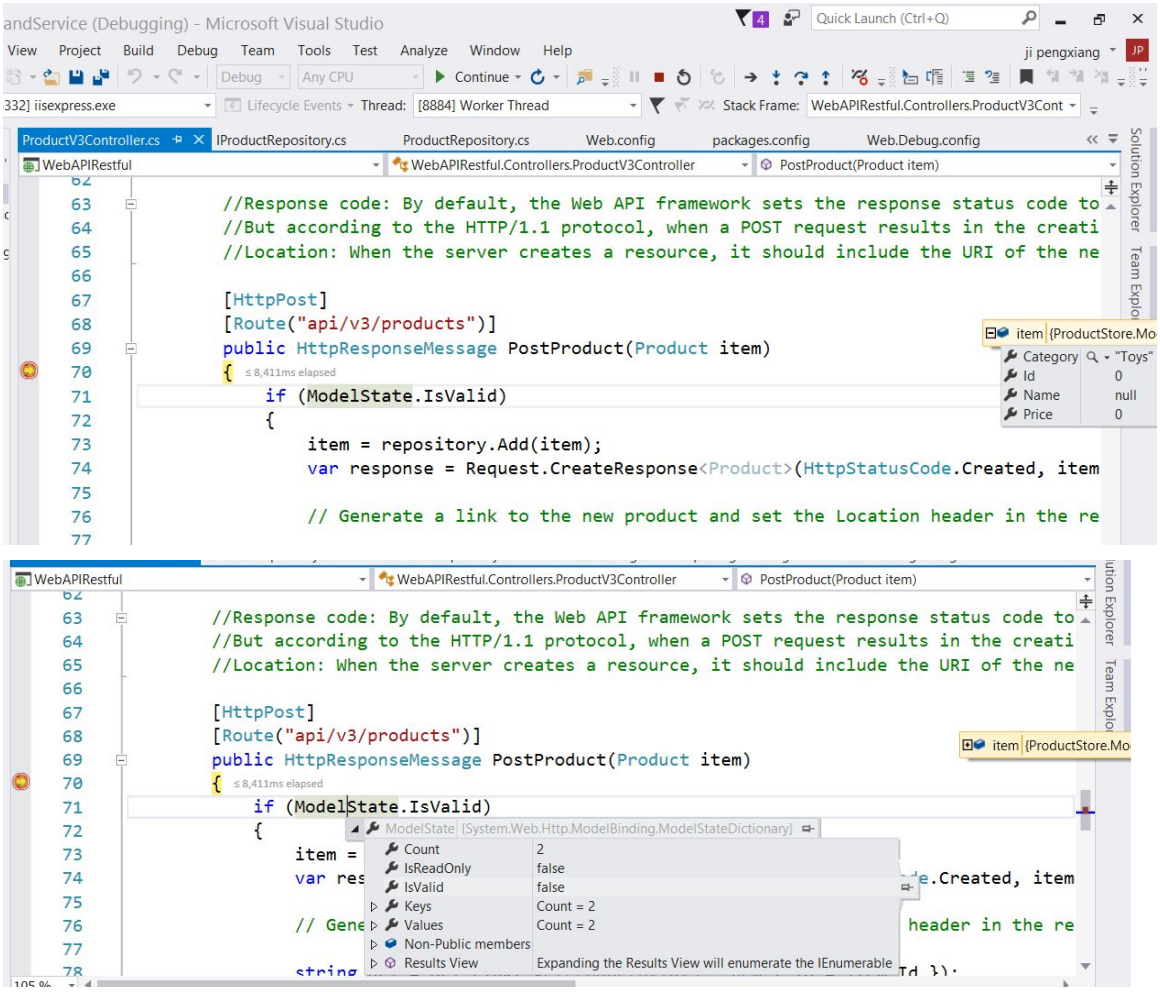
DEBUGGING

Here, the client did not specify values for Name. The JSON formatter assigns a default value of null to the missing properties



{ "Category": "Toys" }

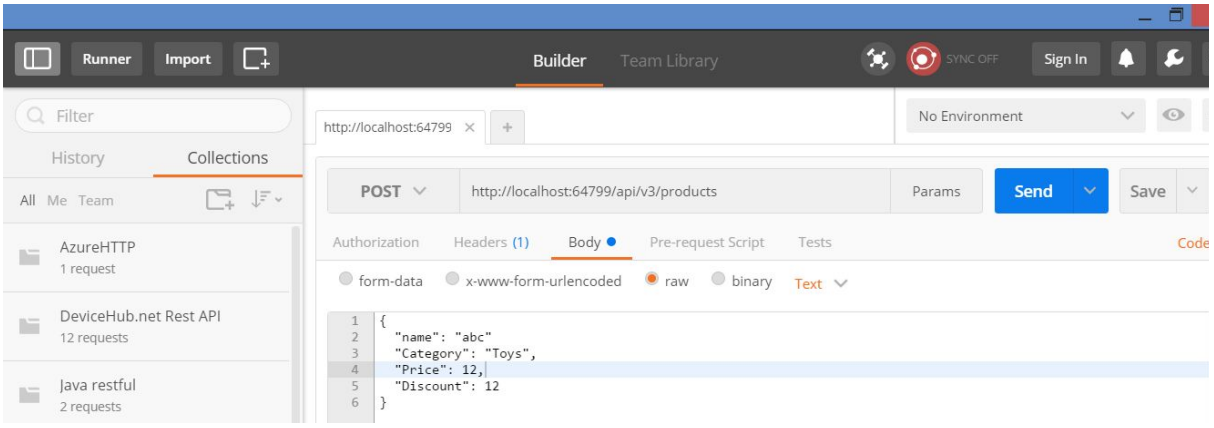
Here, the client did not specify values for `Price`. The JSON formatter assigns a default value of zero to the missing properties

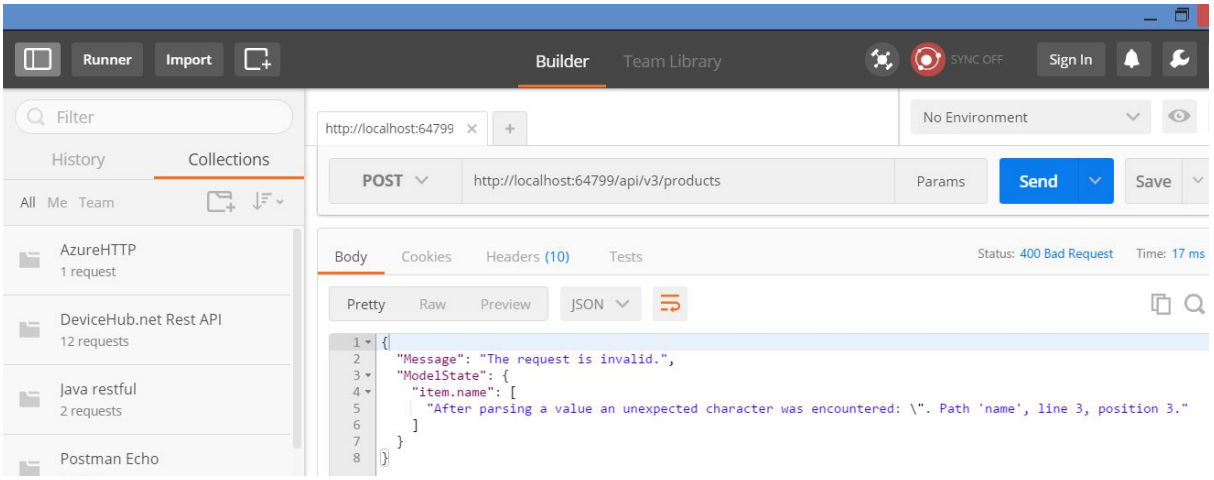


"Over-Posting": A client can also send *more* data than you expected. For example:

```
{ "name": "abc",   "Category": "Toys",   "Price": 12,  "Discount": 12}
```

Here, the JSON includes a property ("Discount") that does not exist in the `Product` model.





(**Optional**)

You can also create an action filter to check the model state before the controller action is invoked. The following code shows an example:

```
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http.Controllers;
using System.Web.Http.Filters;
using System.Web.Http.ModelBinding;

namespace MyApi.Filters
{
    public class ValidateModelAttribute : ActionFilterAttribute
    {
        public override void OnActionExecuting(HttpActionContext actionContext)
        {
            if (actionContext.ModelState.IsValid == false)
            {
                actionContext.Response =
actionContext.Request.CreateErrorResponse(
                    HttpStatusCode.BadRequest, actionContext.ModelState);
            }
        }
    }
}
```

If model validation fails, this filter returns an HTTP response that contains the validation errors. In that case, the controller action is not invoked.

To apply this filter to all Web API controllers, add an instance of the filter to the **HttpConfiguration.Filters** collection during configuration:

```
public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            config.Filters.Add(new ValidateModelAttribute());

            // ...
        }
}
```

Another option is to set the filter as an attribute on individual controllers or controller actions:

```
public class ProductsController : ApiController
{
    [ValidateModel]
    public HttpResponseMessage Post(Product product)
    {
        // ...
    }
}
```

Reference:

`Model Validation in ASP.NET Web API`

https://www.asp.net/web-api/overview/formats-and-model-binding/model-validation-in-aspnet-web-api

ASP.NET Web API Model Validation with Custom Action Filter

http://www.dotnetcurry.com/aspnet/1191/aspnet-web-api-model-validation-custom-action-filter

PART 2

Tasks

1. Type the XML document slowly

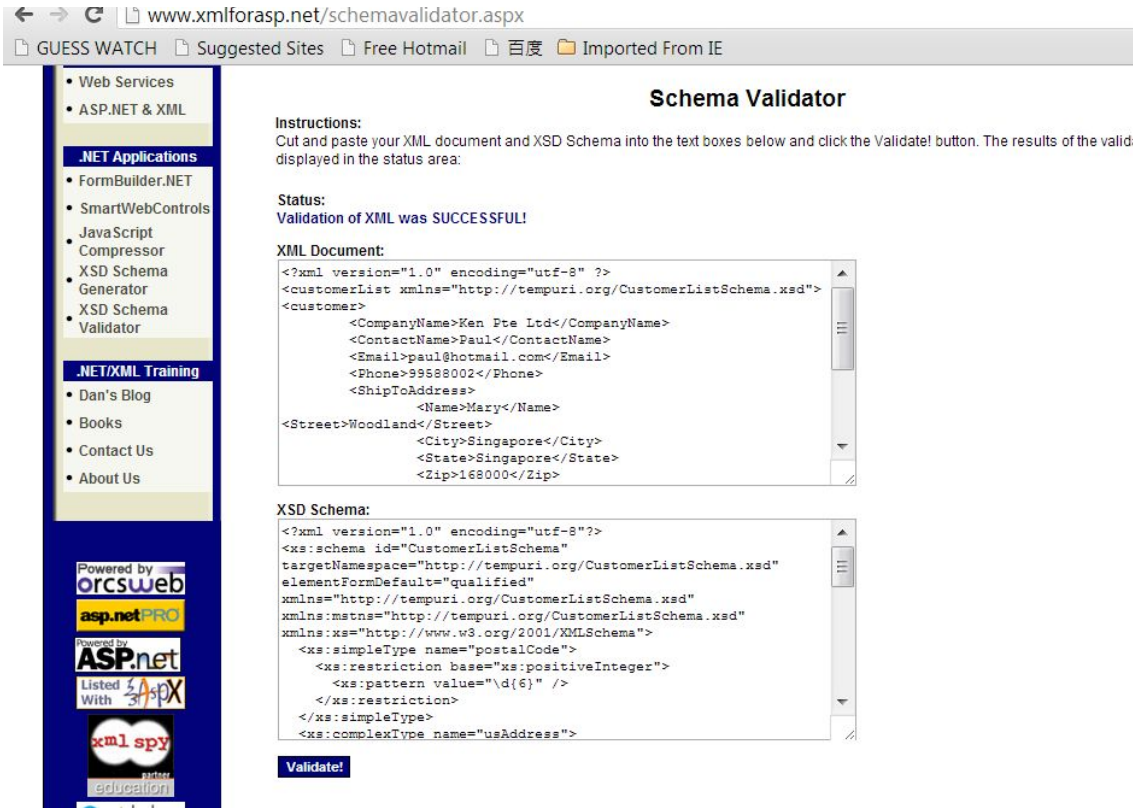
```
<?xml version="1.0" encoding="utf-8" ?>
<customerList xmlns="http://tempuri.org/CustomerListSchema.xsd">
 <Customer>
       <CompanyName>Ken Pte Ltd</CompanyName>
       <ContactName>Paul</ContactName>
       <Email>paulhotmail.com</Email>
       <Phone>99588002</Phone>
      <ShipToAddress>
              <Name>Mary</Name>
<Street>Woodland</Street>
              <City>Singapore</City>
              <State>Singapore</State>
              <Zip>168000</Zip>
       </ShipToAddress>
<BillToAddress>
              <Name>Ken Lim</Name>
              <Street>Bedok</Street>
              <City>Singapore</City>
              <State>Singapore</State>
              <Zip>681006</Zip>
       </BillToAddress>
       </Customer>
  <Customer>
    <CompanyName>Ken Pte Ltd</CompanyName>
    <ContactName>Ji</ContactName>
    <Email>paul@hotmail.com</Email>
    <Phone>99588002</Phone>
    <ShipToAddress>
      <Name>Mary</Name>
      <Street>Woodland</Street>
      <City>Singapore</City>
      <State>Singapore</State>
      <Zip>168000</Zip>
    </ShipToAddress>
    <BillToAddress>
      <Name>Ken Lim</Name>
      <Street>Bedok</Street>
      <City>Singapore</City>
      <State>Singapore</State>
      <Zip>681006</Zip>
    </BillToAddress>
  </Customer>
</customerList>
```

To add a new XML Schema item to the project

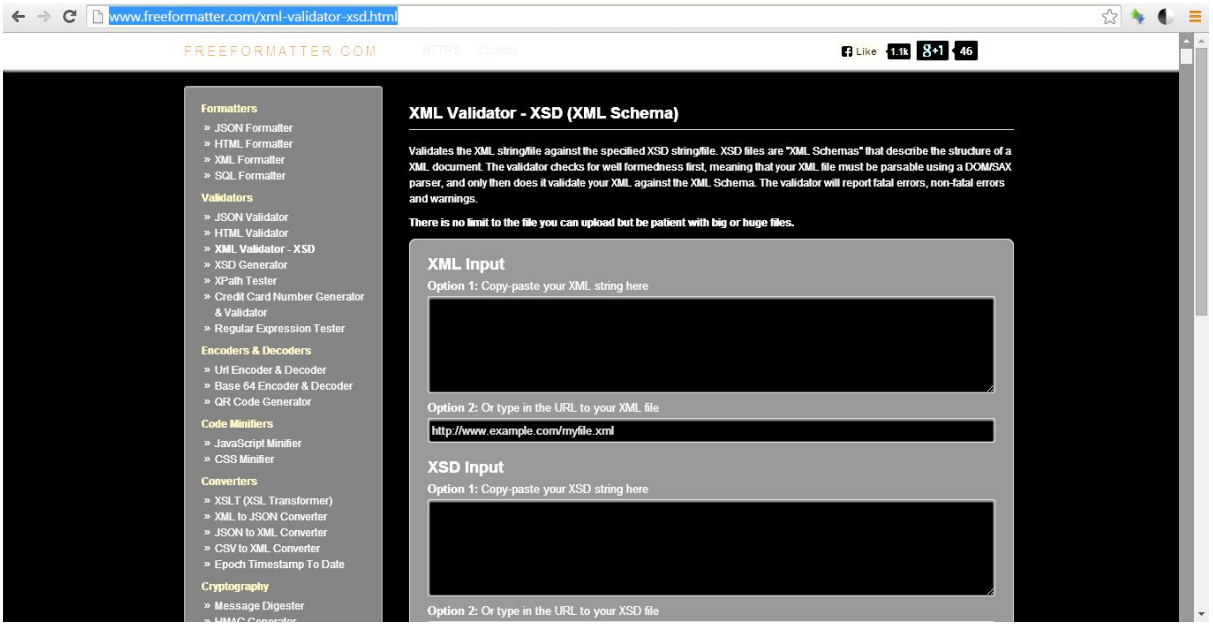
```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="CustomerListSchema"
targetNamespace="http://tempuri.org/CustomerListSchema.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/CustomerListSchema.xsd"
xmlns:mstns="http://tempuri.org/CustomerListSchema.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="postalCode">
    <xs:restriction base="xs:positiveInteger">
      <xs:pattern value="\d{6}" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="usAddress">
    <xs:sequence>
      <xs:element name="Name" type="xs:string" />
      <xs:element name="Street" type="xs:string" />
      <xs:element name="City" type="xs:string" />
      <xs:element name="State" type="xs:string" />
      <xs:element name="Zip" type="postalCode" />
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="Email">
    <xs:restriction base="xs:string">
      <xs:pattern
value="[A-Za-z0-9_]+([-+.'][A-Za-z0-9_]+)*@[A-Za-z0-9_]+([-.][A-Za-z0-9_]+)*\.[A-Za-z0-9
_]+([-.][A-Za-z0-9_]+)*" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="customerList">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Customer" maxOccurs="unbounded" >
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CompanyName" type="xs:string" />
              <xs:element name="ContactName" type="xs:string" />
              <xs:element name="Email" type="Email" />
              <xs:element name="Phone" type="xs:string" />
              <xs:element name="ShipToAddress" type="usAddress" />
              <xs:element name="BillToAddress" type="usAddress" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Validate XML file using online tools (optional)

1. Go to url at http://www.xmlforasp.net/schemavalidator.aspx
2. Copy and paste XML file and Schema file

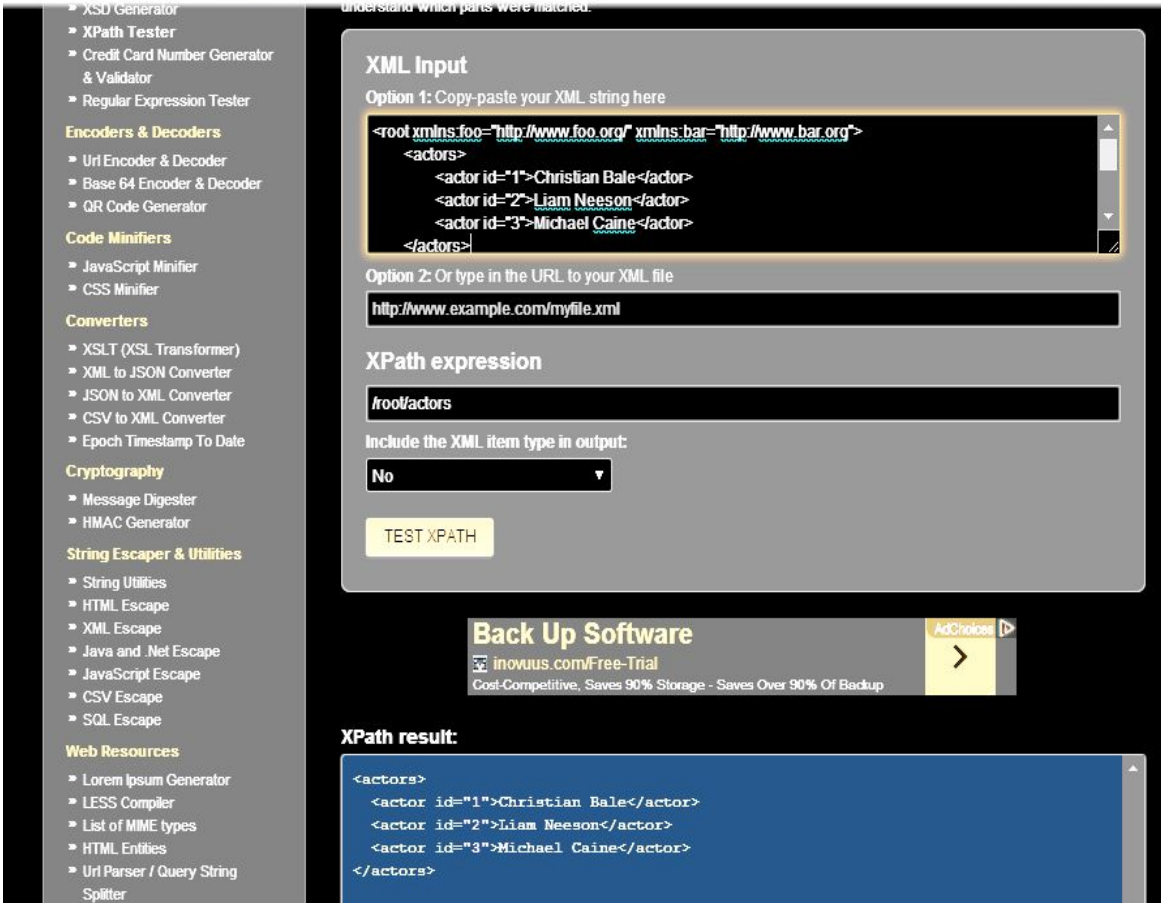


Or you can use the xml validator tools at http://www.freeformatter.com/xml-validator-xsd.html



B. Using online tools to explore Xpath

http://www.freeformatter.com/xpath-tester.html



C. Using XPath Expression to Access or Read XML document in JavaScript

Source: http://www.w3schools.com/xpath/xpath_examples.asp

1. Create a XML file as follows "books.xml":

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

<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">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</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>
```

View the "books.xml" file in your browser.

2. Add a HTML page as bookResultXpath.html

See Code in Action »

```
<!DOCTYPE html>
<html>
<body>
    <script>

        function loadXMLDoc(dname) {
            if (window.XMLHttpRequest) {
                xhttp = new XMLHttpRequest();
            }
            else {
                xhttp = new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET", dname, false);
            try { xhttp.responseType = "msxml-document" } catch (err) { } // Helping IE
            xhttp.send("");
            return xhttp;
        }

        var x = loadXMLDoc("books.xml");
        var xml = x.responseXML;
```

```
        //var path = "/bookstore/book/title";

        //select the title of the first book node under the bookstore element:
        //var path="/bookstore/book[1]/title"

        // select price nodes with price>35
        var path = "/bookstore/book[price>35]/price"

        //select title nodes with price>35
        var path = "/bookstore/book[price>35]/title"

        //select title nodes with price>35
        var path = "//book[price>35]/title"

        // ?Select the 'book' element with the 'category attribute value of 'COOKING'
         //var path = "/bookstore/book[category='COOKING']/title";
        // code for IE
        if (window.ActiveXObject || xhttp.responseType == "msxml-document") {
            xml.setProperty("SelectionLanguage", "XPath");
            nodes = xml.selectNodes(path);
            for (i = 0; i < nodes.length; i++) {
                document.write(nodes[i].childNodes[0].nodeValue);
                document.write("<br>");
            }
        }

            // code for Chrome, Firefox, Opera, etc.
        else if (document.implementation && document.implementation.createDocument) {
            var nodes = xml.evaluate(path, xml, null, XPathResult.ANY_TYPE, null);
            var result = nodes.iterateNext();

            while (result) {
                document.write(result.childNodes[0].nodeValue);
                document.write("<br>");
                result = nodes.iterateNext();
            }
        }

    </script>

</body>
</html>
```

Note: Go to http://www.ajaxload.info/ to generate a ajax loader `ajax-loader.gif`.

Change the above code to display

select the title of the first book node under the bookstore element:
/bookstore/book[1]/title

Select the text from all the price nodes:
/bookstore/book/price[text()]

select price nodes with price>35

/bookstore/book[price>35]/price

select title nodes with price>35

/bookstore/book[price>35]/title

D. Convert XML to JSON …

http://www.freeformatter.com/xml-to-json-converter.html#ad-output

```
{
```

```
  "Customer": [
    {
      "CompanyName": "Ken Pte Ltd",
      "ContactName": "Paul",
      "Email": "paulhotmail.com",
      "Phone": "99588002",
      "ShipToAddress": {
        "Name": "Mary",
        "Street": "Woodland",
        "City": "Singapore",
        "State": "Singapore",
        "Zip": "168000"
      },
      "BillToAddress": {
        "Name": "Ken Lim",
        "Street": "Bedok",
        "City": "Singapore",
        "State": "Singapore",
        "Zip": "681006"
```

```
        }
    },
    {
        "CompanyName": "Ken Pte Ltd",
        "ContactName": "Ji",
        "Email": "paul@hotmail.com",
        "Phone": "99588002",
        "ShipToAddress": {
            "Name": "Mary",
            "Street": "Woodland",
            "City": "Singapore",
            "State": "Singapore",
            "Zip": "168000"
        },
        "BillToAddress": {
            "Name": "Ken Lim",
            "Street": "Bedok",
            "City": "Singapore",
```

```
            "State": "Singapore",

            "Zip": "681006"

        }

    }

  ]

}
```

I. XML Serialization and Deserialization

Serialization is the process of taking the state of an object and persisting it in some fashion. The Microsoft .NET Framework includes powerful objects that can serialize any object to XML. The **System.Xml.Serialization** namespace provides this capability.

Follow these steps to create a console application that creates an object, and then serializes its state to XML:

1. In Visual C#, Open your Console Application Project **ConsoleClient**. Create a new Console Application project if it does not exist.

2. On the **Project** menu, click **Add Class** to add a new class to the project.

3. In the **Add New Item** dialog box, change the name of the class to Person.

4. Click **Add**. A new class is created.

5. Add the following code after the Public Class **Person** statement

```
public   string FirstName;

public   string MI;

public   string LastName;
```

Add a new class **SerilizeObjectXML**. Switch to the code window for SerializeObjectXML.

7. In the void **Main** method, declare and create an instance of the **clsPerson** class:

```
clsPerson p = new clsPerson();
```

8. Set the properties of the **clsPerson** object:

```
p.FirstName = "Jeff";

    p.MI = "A";

        p.LastName = "Price";
```

9. The **Xml.Serialization** namespace contains an **XmlSerializer** class that serializes an object to XML. When you create an instance of **XmlSerializer**, you pass the type of the class that you want to serialize into its constructor:

```
System.Xml.Serialization.XmlSerializer x = new
System.Xml.Serialization.XmlSerializer(p.GetType());
```

10. The **Serialize** method is used to serialize an object to XML. Serialize is overloaded and can send output to a **TextWriter**, **Stream**, or **XMLWriter** object. In this example, you send the output to the console:

```
x.Serialize(Console.Out,p);

   Console.WriteLine();

   Console.ReadLine();
```

Complete Code Listing

```
using System;
using System.Collections.Generic;

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace ConsoleWebAPIClient
{

       class SerializeObjectXML
       {
       static void Main(string[] args)
       {
              Person p = new Person();
              p.FirstName = "Jeff";
              p.MI = "A";
              p.LastName = "Price";
              XmlSerializer x = new XmlSerializer(p.GetType());
              x.Serialize(Console.Out, p);
              Console.WriteLine();
              Console.ReadLine();

              // To write to a file, create a StreamWriter object.
              StreamWriter myWriter = new StreamWriter("myXML.xml");
              x.Serialize(myWriter, p);
              myWriter.Close();
       }


       }
}
```

Verification

To verify that your project works, press CTRL+F5 to run the project. A p**erson** object is created and populated with the values that you entered. This state is serialized to XML. The console window shows the following:

```
<?xml version="1.0" encoding="IBM437"?>
<clsPerson xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3
.org/2001/XMLSchema">
<FirstName>Jeff</FirstName>
<MI>A</MI><LastName>Price</LastName>
</clsPerson>
```

Troubleshoot

The **Xml.Serialization.XmlSerializer** object performs only shallow serialization. If you also want to serialize the private variables of an object or child objects, you must use deep serialization.

Create a new class SerializeObjectsToXML to serialize a list of persons and deserialize.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;

namespace ConsoleWebAPIClient
{

    class SerializeObjectsToXML
    {
        static List<Person> persons = new List<Person>();
        static void Main(string[] args)
        {
            Person p1 = new Person();
            p1.FirstName = "Jeff";
            p1.MI = "A";
            p1.LastName = "Price";
            persons.Add(p1);

            Person p2 = new Person();
            p2.FirstName = "Jeff";
            p2.MI = "A";
            p2.LastName = "Price";
            persons.Add(p2);
            serialize();
            deserialize();
            Console.ReadLine();

        }

        private static void serialize()
        {
            XmlSerializer x = new XmlSerializer(persons.GetType());
            x.Serialize(Console.Out, persons);
            Console.WriteLine();

            // To write to a file, create a StreamWriter object.
            StreamWriter myWriter = new StreamWriter("myXML.xml");
            x.Serialize(myWriter, persons);
            myWriter.Close();
        }




        public static void deserialize()
        {
            XmlSerializer myDeserializer = new XmlSerializer(persons.GetType());
            FileStream myFileStream = new FileStream("myXML.xml", FileMode.Open);
            var listOfPersons = (List<Person>)myDeserializer.Deserialize(myFileStream);
            myFileStream.Close();

            //display the result in the list
            int numofPersons = listOfPersons.Count;

            Console.WriteLine("number of persons" + numofPersons);
            foreach (Person p in listOfPersons)
            {
                Console.WriteLine("{0}\t${1}\t{2}",
```

```
                            p.LastName,
                            p.FirstName,
                            p.MI);
                }
            }
        }
    }
```

J. JSON Serialization and Deserialization

Add a new class to the project **SerializeObjectsToJSON (referenece to Newtonsoft.Json)**

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Web.Script.Serialization;

namespace ConsoleWebAPIClient
{
   class SerializeObjectsToJSON
   {
      static List<Person> persons = new List<Person>();
      static void Main(string[] args)
      {
         Person p1 = new Person();
         p1.FirstName = "Jeff";
         p1.MI = "A";
         p1.LastName = "Price";
         persons.Add(p1);

         Person p2 = new Person();
         p2.FirstName = "Jeff";
         p2.MI = "A";
         p2.LastName = "Price";
         persons.Add(p2);
         serialize();
         deserialize();

         Console.ReadLine();

      }

      private static void serialize()
      {
        // var javaScriptSerializer = new
   //JavaScriptSerializer();
//string jsonString = javaScriptSerializer.Serialize(persons);

//higher performance

string jsonString = JsonConvert.SerializeObject(persons, Formatting.Indented);
         Console.WriteLine(jsonString);

         // To write to a file, create a StreamWriter object.
         StreamWriter myWriter = new StreamWriter("myJSON.json");
         myWriter.WriteAsync(jsonString);
```

```
            myWriter.Close();

        }

        public static void deserialize()
        {
          // JavaScriptSerializer js = new JavaScriptSerializer();
            string jsonString = File.ReadAllText("myJSON.json");
            //List<Person> persons = js.Deserialize<List<Person>>(jsonString);
            List<Person> persons = JsonConvert.DeserializeObject<List<Person>>(jsonString);


            //display the result in the list
            int numofPersons = persons.Count;

            Console.WriteLine("number of persons" + numofPersons);
            foreach (Person p in persons)
            {
               Console.WriteLine("{0}\t${1}\t{2}",
                        p.LastName,
                        p.FirstName,
                        p.MI);
            }
        }

    }
}
```

Troubleshoot

Json.NET is a popular high-performance JSON framework for .NET

To install Json.NET, run the following command in the Package Manager Console

```
Install-Package Newtonsoft.Json
```

Output for reference:

D. Tutorial Question for you to try in your own time.

Using XPath Expression to Access or Read XML document in C# ASP.Net

XML is one of the widely used data exchange format for variety of reasons in software industry. Hence, working on XML data like selecting a XML node, navigating the XML node tree for finding a node or doing some manipulation on XML nodes has become one of the major and repetitive tasks in any project we work. In this article, we will try to understand how to read XML data using XPath expression in ASP.Net.

What is XPath expression?

Understanding the increasing need of XML, W3C has developed a new querying language calledXPath expression through which we can easily walk into a XML document and select a node. In simple terms, XPath is simple and easily understandable expressions which we can use to select XML node or nodes in an XML document.

```
<?xml version="1.0" encoding="utf-8"?>
<Employees>
  <Employee type="Permanent">
    <ID>100</ID>
    <FirstName>Bala</FirstName>
    <LastName>Murugan</LastName>
    <Dept>Production Support</Dept>
  </Employee>
  <Employee type="Contract">
    <ID>101</ID>
    <FirstName>Peter</FirstName>
    <LastName>Laurence</LastName>
    <Dept>Development</Dept>
  </Employee>
  <Employee type="Permanent">
    <ID>102</ID>
    <FirstName>Rick</FirstName>
    <LastName>Anderson</LastName>
    <Dept>Sales</Dept>
  </Employee>
  <Employee type="Contract">
    <ID>103</ID>
    <FirstName>Ramesh</FirstName>
    <LastName>Kumar</LastName>
    <Dept>HR</Dept>
  </Employee>
  <Employee type="Permanent">
    <ID>104</ID>
    <FirstName>Katie</FirstName>
    <LastName>Yu</LastName>
    <Dept>Recruitment</Dept>
  </Employee>
  <Employee type="Contract">
    <ID>105</ID>
    <FirstName>Suresh</FirstName>
    <LastName>Babu</LastName>
    <Dept>Inventory</Dept>
  </Employee>
</Employees>
```

XPath expression Syntax

The XPath expression will use / to select the root node of the XML document. For example, to select all the employee nodes in the above XML we have to use the below XPath query,

```
/Employees/Employee
```

The above XPath query will return all nodes under every <Employee> nodes.

The below table list some of the important expressions to build an XPath query.

Expression	Description
node	To select all child nodes under the node "node"
/	To select the root node
//	To select the node from any where in the document.
.	To select the current node
..	To select the parent of the current node
@	To select attributes

Moving forward, we will see how these XPath expressions can be used in .net world to access and manipulate XML contents.

In this section, we will see how to read a XML node at a very basic level using XmlDocument class System.Xml namespace. We can use SelectSingleNode() and SelectNodes() methods of XmlDocument class to query the XML using XPath expression. See below,

To select a single node, for example first name,

```
XmlDocument xmldoc = new XmlDocument();
xmldoc.Load(Server.MapPath("~/App_Data/Employees.xml"));
Response.Write(xmldoc.SelectSingleNode("/Employees/Employee/FirstName").InnerText);
```

OUTPUT

```
Bala
```

To select nth occurence of a node value, for example to get 3rd employee's firstname,

```
XmlDocument xmldoc = new XmlDocument();
xmldoc.Load(Server.MapPath("~/App_Data/Employees.xml"));
Response.Write(xmldoc.SelectSingleNode("/Employees/Employee[3]/FirstName").InnerTex
t);
```

OUTPUT

```
Rick
```

To select a node conditionally, for example to select firstname of a employee with ID as 101,

```
XmlDocument xmldoc = new XmlDocument();
xmldoc.Load(Server.MapPath("~/App_Data/Employees.xml"));
Response.Write(xmldoc.SelectSingleNode("/Employees/Employee[ID=101]/FirstName").Inn
erText);
```

OUTPUT

```
Peter
```

To select a attribute of a XML node, for example to get the type of employee with ID 101,

```
XmlDocument xmldoc = new XmlDocument();
xmldoc.Load(Server.MapPath("~/App_Data/Employees.xml"));
 Response.Write(xmldoc.SelectSingleNode("/Employees/Employee[ID=101]/@type").InnerT
ext);
```

OUTPUT

```
Contract
```

To get a count of all permanent employees,

```
XmlDocument xmldoc = new XmlDocument();
xmldoc.Load(Server.MapPath("~/App_Data/Employees.xml"));
Response.Write(xmldoc.SelectNodes("/Employees/Employee[@type='Permanent']").Count.T
oString());
```

OUTPUT

```
3
```

To get all employee count,

```
XmlDocument xmldoc = new XmlDocument();
xmldoc.Load(Server.MapPath("~/App_Data/Employees.xml"));
Response.Write(xmldoc.SelectNodes("//Employee").Count.ToString());
```

OUTPUT

```
6
```

At times, we need to fetch some set of XML nodes to populate an ASP.Net control. The below code will fetch all permanent employees, contract employees, all employees and populate them to a ddlPermanentEmp, ddlContractEmp, ddlAllEmployees DropDownList respectively.

```
XmlDocument xmldoc = new XmlDocument();
xmldoc.Load(Server.MapPath("~/App_Data/Employees.xml"));
//Get permanent employees
  XmlNodeList nodes = xmldoc.SelectNodes("/Employees/Employee[@type='Permanent']");
        foreach (XmlNode node in nodes)
        {
            ddlPermanentEmp.Items.Add(new
ListItem(node.SelectSingleNode("FirstName").InnerText,
node.SelectSingleNode("ID").InnerText));
        }
//Get contract employees
nodes = xmldoc.SelectNodes("/Employees/Employee[@type='Contract']");
        foreach (XmlNode node in nodes)
        {
            ddlContractEmp.Items.Add(new
ListItem(node.SelectSingleNode("FirstName").InnerText,
node.SelectSingleNode("ID").InnerText));
        }

//Get all employees
        nodes = xmldoc.SelectNodes("//Employee");
        foreach (XmlNode node in nodes)
        {
            ddlAllEmployees.Items.Add(new
ListItem(node.SelectSingleNode("FirstName").InnerText,
node.SelectSingleNode("ID").InnerText));
        }
```

Thus, with the above examples we have seen some of basic usages of XPath expressions to query the XML content.

Thus, with the above examples we have seen some of basic usages of XPath expressions to query the XML content.
In the coming section, we will see the usages of XPath related classes in System.Xml.XPath namespace in .NetFramework.

XPath support in .NetFramework

The System.Xml.XPath namespace has a class called XPathNavigator which can be used for both forward and backward navigation of XML nodes. Technically, it is an abstract

class which defines a cursor model for navigating and editing XML information items as instances of the XPath 2.0 Data Model. Once after selecting the XML nodes using an XPath expression, it uses XPathNodeIterator class to iterate through the XML nodes. To create XPathNavigator object, we need to call the CreateNavigator() method on XmlDocument or XpathDocument object. The below code will read all permanent employees and load it to a DropDownList ddlEmployees.

```
        XPathDocument xpathdoc = new
XPathDocument(Server.MapPath("~/App_Data/Employees.xml"));
        XPathNavigator navigator = xpathdoc.CreateNavigator();
        XPathExpression expr = navigator.Compile("/Employees/Employee[@type='Permanent']");
        XPathNodeIterator nodes = navigator.Select(expr);

        while (nodes.MoveNext())
        {
            ddlEmployees.Items.Add(new
ListItem(nodes.Current.SelectSingleNode("FirstName").Value,
nodes.Current.SelectSingleNode("ID").Value));
        }
```

Source:

http://www.codedigest.com/Articles/ASPNET/342_Using_XPath_Expression_to_Access_or_Read_XML_document_in_ASPNet.aspx

Serializing_Collections

Json.NET has excellent support for serializing and deserializing collections of objects.

Serializing Collections

To serialize a collection - a generic list, array, dictionary, or your own custom collection - simply call the serializer with the object you want to get JSON for. Json.NET will serialize the collection and all of the values it contains.

Serializing Collections

```
Product p1 = new Product
 {
     Name = "Product 1",
     Price = 99.95m,
     ExpiryDate = new DateTime(2000, 12, 29, 0,
0, 0, DateTimeKind.Utc),
 };
 Product p2 = new Product
 {
     Name = "Product 2",
    Price = 12.50m,
    ExpiryDate = new DateTime(2009, 7, 31, 0, 0,
0, DateTimeKind.Utc),
};

List<Product> products = new List<Product>();
products.Add(p1);
products.Add(p2);

string json =
JsonConvert.SerializeObject(products,
Formatting.Indented);
19//[
20//  {
21//    "Name": "Product 1",
22//    "ExpiryDate": "2000-12-29T00:00Z",
23//    "Price": 99.95,
24//    "Sizes": null
25//  },
26//  {
27//    "Name": "Product 2",
```

```
//    "ExpiryDate": "2009-07-31T00:00Z",
//    "Price": 12.50,
//    "Sizes": null
//  }
//]
```

Deserializing Collections

To deserialize JSON into a .NET collection, just specify the collection type you want to deserialize to. Json.NET supports a wide range of collection types.

Deserializing Collections

```
string json = @"[
   {
     'Name': 'Product 1',
     'ExpiryDate': '2000-12-29T00:00Z',
     'Price': 99.95,
     'Sizes': null
   },
   {
     'Name': 'Product 2',
    'ExpiryDate': '2009-07-31T00:00Z',
    'Price': 12.50,
    'Sizes': null
  }
]";

List<Product> products =
JsonConvert.DeserializeObject<List<Product>>(jso
n);

Console.WriteLine(products.Count);

Product p1 = products[0];

Console.WriteLine(p1.Name);
// Product 1
```

Deserializing Dictionaries

Using Json.NET you can also deserialize a JSON object into a .NET generic dictionary. The JSON object's property names and values will be added to the dictionary.

Deserializing Dictionaries

```
string json =
@"{""key1"":""value1"",""key2"":""value2""}";

Dictionary<string, string> values =
JsonConvert.DeserializeObject<Dictionary<string,
string>>(json);

Console.WriteLine(values.Count);
// 2

Console.WriteLine(values["key1"]);
// value1
```