Unit 5 – Working with XML, ASP.NET Application Configuration and Deployment of Application

Reading DataSets with XML

In order to read data from an XML file to a DataSet, a method called “**ReadXML()**” of DataSet object can be used.
This method takes the path of an XML file as its only argument.
There is another method called “ReadXMLSchema()” which reads only the schema.
In the following example, data from a XML file is read using ReadXML() method of DataSet object. The data of the DataSet is then displayed in a GridView control.

Example :
```
using System;
using System.Data.SqlClient;
public partial class Default2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        DataSet ds = new DataSet();
        ds.ReadXml("d:/xyz/empdata.xml");
        GridView1.DataSource = ds;
        GridView1.DataBind();
    }
}
```

Writing DataSets to XML

In order to write the records of a DataSet in an XML file, a method called “**WriteXML()**” of DataSet object can be used.
The WriteXML() method not only writes the data of a given dataset in XML format, but it also creates the XML file in which data is written.
The WriteXML() method takes one argument and it is the path of the XML file.
Another method of DataSet object is “WriteXMLSchema()”. This method writes only the schema to the XML file.
The following example writes the data of “emp” table of “company” database in an XML file called “empdata.xml”.

Example :
```
using System;
using System.Data.SqlClient;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
```

```
    SqlConnection con = new SqlConnection("Data Source=ABC;Initial
    Catalog=company;Integrated Security=True;Pooling=False");
    SqlDataAdapter da = new SqlDataAdapter("select * from emp", con);
    DataSet ds = new DataSet();
    da.Fill(ds);
    ds.WriteXml("d:/xyz/empdata.xml");
    }
}
```

Introduction to WebServices

"**A WebService is an entity that can be programmed to provide a particular functionality to any number of applications over the Internet.**"
The functionality provided by the WebService can be used by an application of any programming language and in which it is created or the hardware and operating system on which it is running.
A WebSevice does not have any user interface. A WebService just provide specific services to its consumers.
Before WebServices where invented, there were other Component based technologies used such as **DCOM (Distributed Component Object Model) and CORBA (Common Object Request Broker Architecture).**
WebServices became popular because of the advantages they provide over other component based technologies.
The following are the advantages of WebServices :

1) WebServices are simple because of which they can be implemented on different platforms.
2) WebServices are loosely coupled because of which their interfaces and methods can be extended.
3) WebServices don't carry any state information with them.
4) WebServices are not affected by Firewalls.

WebServices are mostly used while communicating across cross-platform or enforcing trusted connections between server and the client.
WebServices can be accessed by any application, independent of the software and hardware platform on which the application is running because it compiles with common industry standards, such as **SOAP (Simple Object Access Protocol) and WSDL (WebServices Description Language).**

SOAP

SOAP stands for **Simple Object Access Protocol**.
SOAP allows webservice to communicate with other web application over the internet. SOAP specifies a format for the XML message using which the web application request and receive webservice from a web server. SOAP messages are transmitted across the

web application using the HTTP protocol. This is because HTTP is a standard protocol which is supported by almost every operating system.
The two features that make SOAP very popular protocol are as follows :

1) Support for **Remote Procedure Calls (RPC)**.
2) Platform independency.

In RPC, a remote web application sends a SOAP message that contains a request to a procedure of the webservice.
The SOAP message also contains the parameters that are to be passed to the procedure while accessing it. When the web server running the webservice receives the message, it sends the result of the procedure to the web application.
The platform independent feature of SOAP allows a webservice to communicate with various applications, irrespective of the hardware and software platform on which they are running.

Creating (Building) & Consuming (using) a WebService

Creating a WebService:
The following are the steps for creating a new webservice:

1) Create a new website application in Visual Studio.
2) Select the "**Add New Item**" from the Website menu.
3) Select the "**Web Service**" option from the available templates (file types).
4) Save this webservice by giving a proper name to it. The file will have **.asmx** extension.
5) Click the "**Add**" button.
6) To write the code which gives functionality to the webservice, open the code window of webservice file.
7) Change the name (text) of the namespace according to your solution.
8) Write the code for "**Web Method**" as per the requirement.

Consuming a WebService:
In order to consume (use) this already created webservice, a reference must be added. The following are its steps :

1) Right-Click on "service.asmx" file (if the file is saved with the name "service.asmx") and click the "Build Page" option. This step builds the webservice and if it is successfully build then follow the next step.
2) Right-Click on the solution name and select "**Add Service Reference**" option.
3) Click on "**Advanced**" button.
4) Click on "**Add Web Reference**" button.
5) In the dialog box you can enter a URL of any webservice of any site if you are connected to the internet. Here, to add the webservice created, click the "Web services in this solution" link.
6) Click on the web service name (here, "service") link given below "Services" column.

7) Finally, click the "**Add Reference**" button.

Now to use this webservice in your web application, you can follow these steps :

1) Add a Webform to your application.
2) Take controls on the form as per requirement.
3) Write the necessary code to use the webservice. For example,

```
localhost.service ls = new localhost.service();
Label3.Text = ls.sum(x,y).ToString();
```

The above code uses a webservice having a web method called "sum()" which calculates the sum of two numbers entered in 2 textboxes and displays its result in a label.

HTTP

HTTP stands for **Hyper Text Transfer Protocol.**

HTTP is a set of rules that manages the transfer of Hyper Text between two or more computers.

Hyper Text is a text that is coded using Hyper Text Markup Language (HTML). The HTML code is sued to create hyperlinks. This links can be textual or graphical.

HTTP is based on client-server principle. It allows the client to create a connection to the server and make a request. The server accepts the request of the client and sends back a response.

Basically, HTTP is a TCP/IP based communication protocol which is used to give data (HTML files, image files, query results etc) on the World Wide Web (www).

HTTP provides a standardized way for computers to communicate with each other.

The following are the features of HTTP :

1) HTTP is a connection-less protocol.
2) HTTP is media independent.
3) HTTP is a state-less protocol.

XML

XML stands for **Extensible Markup Language**.

XML is designed to transfer and store data and not to display data. Syntax of XML is much similar to HTML. The main difference between XML and HTML is that XML is designed to **describe** data while HTML is designed to **display** data.

In XML, there are user define tags. In other words, we can define our own tags in XML as per the data. The basic use of XML is **data exchange**.

XML is a platform independent language. XML is a W3C recommendation. Almost all programming languages support XML.

You require an "XML Parser" to run an XML file. An XML Parser is the software that check coding of XML file (its tags, attributes etc) is proper or not. All modern browsers have a built-in XML Parser. XML Parsers converts XML data and displays in the browser. Common XML Parsers are MSXML, REXML, fastXML etc.

UDDI

UDDI stands for **Universal Description, Discovery and Integration**.
"UDDI is an XML based standard for describing, publishing and finding webservices."
UDDI is a specification for distributed registry of webservices. UDDI is a platform independent, open framework.
UDDI can communicate via SOAP, CORBA, JAVA RMI protocols. UDDI uses WSDL to describe webservices. UDDI is seen with SOAP and WSDL as one of three foundation standards of webservices.
Webservice is an open industry initiative enabling businesses to discover each other and define how they interact over the internet.
UDDI has two parts –

1) A registry of all webservice's metadata including a pointer to the WSDL description of a service.
2) A set of WSDL port type definitions for using and searching that registry.

UDDI 1.0 was originally created by Microsoft, IBM and Ariba in September 2000. Currently, UDDI is sponsored by OASIS.

WSDL

WSDL stands for **WebServices Description Language**.
WSDL allows the users of a web service to know about the webservice before using it. WSDL describes all the information required to use a webservice.
The following information is described by WSDL :

1) **Operations** : They refer to the mechanism used by the webservice to exchange messages with the users of the webservice.
2) **Bindings** : They refer to the protocol used by the webservice to interact with the users of webservice.
3) **Endpoints** : They refer to the location of bindings.

WSDL describes a webservice with the help of XML. So, you can use WSDL to describe a webservice that can be used by the application created using different platforms and programming languages. You can read and write WSDL because of XML. A document in which webservice is described is called **WSDL document**.

Introduction to Web.Config

Each and every ASP.NET application has its own copy of the configuration settings stored in a file called Web.config. Web.config is the XML based text file. It is the central location for storing the information to be accessed by the web pages.

Initially web.config file is generated from Machine.config file. Initially web.config file is stored in –

C:\Windows\Microsoft.NET\Framework\<version>\Config

folder of your machine. Then a copy of web.config is kept in your web application folder. If your web application has multiple folders, each folder can have own different set of web.config file.

You can edit web.config file by using standard text editors like Notepad, the ASP.NET snap-in tool, the web site administration tool or the ASP.NET configuration API.

Web.config file contains application wise data like custom settings and error messages, database connection string, user authentication and authorization settings, web page language options, debug options etc.

The root element of web.config is <configuration>. The web.config file is divided into different sections.

Common Configuration Sections

The web.config file is divided into many different configuration sections. As web.config file is XML file, it also has a root element called **<configuration>**.

<configuration> element consists of many other sub elements which are as follows:

1) **<configSections>** : This allows you to configure different sections of web application like JavaScript, web services etc.
2) <**appSettings**> : This allows you to add any application related variable or settings. You can <add>, <remove> or <clear> any application specific variable or settings.
3) <**connectionStrings**> : This allows you to create connection string variables which can be used in any web form of your web application.
4) **<system.web>** : This is one of the important element used to configure your web application. This section allows you to do most of configuration like –
    a. <compilation>
    b. <caching>
    c. <deployment>
    d. <trace>
    e. <webServices>
    f. <authentication>
    g. <customErrors>
    h. <roleManager>
    i. <sessionState>

5) **<system.codedom>** : This section stores compiler specific information used for your web application under <compilers> element.
6) **<system.webServer>** : This section stores information and configuration settings for web servers which will be used for deploying and running your web application.

7) **<runtime>** : This section is used to specify some run time settings for your web application. Run time settings can include number of settings like connection strings, tracing, data providers etc.

appSettings

<appSettings> is one of the sections of configuration settings in the web.config file. It is the sub element of the root element called <configuration> of web.config file.
The <appSettings> section provides a way to define custom application settings for an ASP.NET application. It allows you to add any application related variable or settings. You can <add>, <remove> or <clear> any application specific variable or settings.
Example of <appSettings> is as follows –
Consider two labels on the web form that displays the company name and its owner name. These two can be set using <appSettings>.

The following code is of web.config file –

```
<configuration>
        <appSettings>
                <add key="CompName" value="Abc industries ltd"/>
                <add key="OwnName" value="abc"/>
        </appSettings>
```

The following code is of c# view of the web form –

```
Protected void Page_Load (object sender, EventArgs e)
{
lblcompanyname.Text= ConfigurationManager.AppSettings["CompName"];
lblowner.Text= ConfigurationManager.AppSettings["OwnName"];
}
```

Tracing

ASP.NET provides a tracing feature that enables you to see information related to a request for a web page.
You can also use tracing feature to write debug statements in the application code related to web application.
To allow tracing information to be obtained or displayed, you need to enable tracing for a page or whole application.
You need to set trace attribute in the page directive to true to enable tracing for a page. This process of setting **<trace>** attribute at each webpage is known as **page level tracing.** It allows you to write debugging statements in .aspx page. Page level tracing also allows you to specify the condition on the occurrence of which the debugging statements should

be executed. For this you need to enable the tracing feature in the application by adding following line in the code –

<%@ Page Trace=”true” %>

ASP.NET provides a **Trace class**. This class has a property called “TraceContext”, that you can use to write the debugging statements.
There are two important methods in the Trace class –

1) Write method
2) Warn method

You can use these methods to display a message when an error occurs. The only difference between the two methods is that the output of Warn method is in red colour.
The following is the syntax of the methods –

1) Trace.Write(message)
2) Trace.Warn(Category, Message)

System.Diagnostics.Trace class provides many properties and methods that help in tracing the execution of code in a web application.

Custom Errors

To protect your users from the technical details of the error, you should provide them with a user friendly error page.
The error page should be given in the same style used in your application. ASP.NET allows you to define custom error pages that are shown to the user when an error occurs. You can define the error page you want to show in the **<CustomErrors>** element in the **web.config** file.

For example,

```
<CustomErrors mode=”On” defaultRedirect=”~/Errors/Error500.aspx”>
<error statuscode=”404” redirect=”~/Errrors/Error404.aspx”/>
</CustomErrors>
```

The mode attribute determines whether a visitor to your site gets to see a detail error page or not. It has 3 values –

1) On : Every visitor always sees the custom error pages when an error occurs.
2) Off : The custom error page is never shown.
3) RemoteOnly : With this settings the full error details are shown to the local user while all other users get to see custom error pages.

Within the opening and closing tags of <CustomErrors> element, you have to define separate <error> element. In the code shown above the error is given for the code 404 which is http status code.
When the error is found the given page is displayed. For all other status code, you can use default redirect attribute to show the custom error page.

Although custom error pages protect your users from the error details, they don't help in informing that an error occurred. All this pages do is hide the real error and show a page with a custom error message instead.

Authentication & Authorization

Authentication & Authorization are related to each other.
"Authentication means validating the user name and password of a web application."
If the user has a valid user name and password for accessing the web application, the user is called an Authenticated user.
After the authentication completes, authorization starts.
"Authorization is to determine whether the Authenticated user has the permission or rights to use the resource (web application or database)."

There are 2 types of Authentication –

1) Windows based Authentication
2) Forms based Authentication

1) **Windows based Authentication** : Before a web application checks the records of the user requesting the web application, the IIS (Internet Information Services) checks the user records using its own Authentication methods. Windows based authentication of a web application uses these authentication methods to check the record of the user requesting the web application. Windows based authentication is implemented using "Windows Authentication module" . This provider authenticates a user by creating a "Windows Identify object".

2) **Forms based Authentication** : In Form based application, a web application checks the record of the user, who tries to access the web form called "login form". The login form is used to get and check the username and password. Whenever an authenticated user tries to access the web application, the requests are redirected to the login page. This prevents the user from using the web application if the user does not have a valid username or password.

User & Role Authorization : User and Role authorization enables a web application to allow or restrict (deny) a particular user or role such as Administrator or Database Manager from accessing a particular resource.
To configure user and role authorization, you have to set <**allow**> and <**deny**> attributes in the configuration file of the web application. Both these attributes are defined under <**authorization**>.
The <allow> attribute is used to grant permission whereas <deny> attribute restricts the user or role from accessing a resource.

Deployment of Application in web server

When you have developed, tested and debugged an application, the final step is to deploy it. The purpose of deploying an application is to make sure a simple and easy installation for application files and other required files.
The .NET framework provides the following ways to deploy a web application :

1) Copying the web site contents to the destination.
2) Publishing the application.
3) Using Web Installer.

In the first method the actual source code is copied to the destination so it does not provide any protection to your source code.

The second method provides protection to some limit as it does pre-compilation of source code before sending it to the destination.

The third method provides maximum protection to your source code as it enables you to make fully configurable setup projects for your web application, which are send to the destination instead of the actual source code of application.

Deploying ASP.NET web application using MSI Installer (web installer)

The MSI installer is one of the favorite option available in Microsoft Visual Studio, which can be used to deploy any web site you have created.
The following are the steps to use MSI installer -

1) Open the web site that you want to deploy.
2) Click "**File**" menu -> "**Add**" -> "**New Project**". The "Add New Project" dialog box appears.
3) Select "**Other Project Types**" -> "**Setup and Deployment**" from the left project types part. Select the "**Web Setup Project**" template from the right templates part.
4) Click the "**OK**" button.

This opens the IDE which provides a setup project creations environment known as "File system editor" and it adds "WebSetup1" project in the solution explorer window.

5) Click "**Project**" menu -> "**Add**" -> "**Project Output**". This opens "Add Project Output Group" dialog box.
6) Select "**Configuration**" as "**Debug.NET**" in the dialog box and click "**OK**" button.
7) Click "**Build**" menu -> "**Configuration Manager**". It opens configuration manager dialog box.
8) Select the check box under the "Build" column for "WebSetup1"project and click the close button.
9) Finally click the "**Build**" menu -> "**Build Solution**" option.

The Build process compiles the web site and also adds the output to "WebSetup1" project. The output of this process generates MSI based installer (setup.exe) in the Debug folder of "WebSetup1" project folder.