2PGDCA4 (B)-Programming With ASP. Net

UNIT-V

1. Introduction to XML in .Net

XML is a **cross-platform, hardware and software independent,** text based markup language, which enables you to store data in a structured format by using meaningful tags. XML stores structured data in XML documents that are similar to databases. Notice that **unlike Databases**, XML documents store data in the form of plain text, which can be used across platforms.

In an XML document, you specify the structure of the data by creating a DTD or an XML schema. When you include a DTD in an XML document, the software checks the structure of the XML document against the DTD. This process of checking the structure of the XML document is called ***validating***. The software performing the task of validating is called a validating parser.

The following code defines the structure of an XML document that will store data related to books:

```
<?xml version="1.0"?>
<Books>
 <Book bid="B001">
  <Title> Understanding XML </Title>
  <Price> $30 </Price>
  <author>
   <FirstName> Lily </FirstName>
   <LastName>
    Hicks <LastName>
    </author>
 </Book>
 <Book bid="B002">
  <Title> .NET Framework </Title>
  <Price> $45 </Price>
  <author>
   <FirstName> Jasmine </FirstName>
   <LastName>
    Williams <LastName>
    </author>
 </Book>
</Books>
```

.NET Support for XML

The .NET Framework has extensive support for working with XML documents. IN the .NET framework, the support for XML documents includes:

- ***XML namespace***
- ***XML designer***
- ***XML Web Server control***
- ***XML DOM support***

1.1 XML Namespace

The System.Xml namespace provides a rich set of classes for processing XML data. The commonly used classes for working with XML data are:

- **XmlTextReader**: Provides forward only access to a stream of XML data and checks whether or not an XML document is well formed. This class neither creates as in-memory structure nor validates the XML document against the DTD. You can declare an object of the XmlTextReader class by including the System.Xml namespace in the application. The syntax to declare an object of this class is as follows:

  ```
  XmlTextReader reader = new XmlTextReader("XML1.xml");
  ```

  It is important to note that the .xml file you pass as an argument to the constructor of the XmlTextReader class exists in the \WINNT\System32 folder.

- **XmlTextWriter:** Provides forward only way of generating streams or files containing XML data that conforms to W3C XML 1.0. If you want to declare an object of the XmlTextWriter class, you must include the System.Xml. The syntax to decare an object of this class is as follows:

  ```
  XmlTextWriter writer = new XmlTextWriter(Response.Output);
  ```

  Here Response.Output represents an outgoing HTTP response stream to which you want to send the XML data.

- **XmlDocument:** Provides navigating and edinting features of the nodes in an XML document tree. XmlDocument is the most frequently used class in ASP.NET applications that use XML documents. It also supports W3C XML DOM. *XML DOM is an in-memory representation of an XML document*. It represents data in the form of hierarchically organized object nodes and allows you to programmatically access and manipulate the elements and attributes present in an XML document.

  ```
  XmlDocument doc = new XmlDocument();
  ```

- **XmlDataDocument:** Provides support for XML and relational data in W3C XML DOM. You can use this class with a dataset to provide relational and non-relational views of the same set of data. This class is primarily used when you want to access the functions of ADO.NET. The syntax to declare an object of this class is as follows:

```
DataSet ds=new DataSet();

XmlDataDocument doc=new XmlDocument(ds);
```

There are a number of reasons to use XmlDataDocument:

- It gives you the freedom to work with any data by using the DOM.
- There is synchronization between an XmlDatadocument and a DataSet, and any changes in one will be reflected in the other.
- When an XML document is loaded into an XmlDatadocument, the schema is preserved. You need to include System.Xml namespace.

- **XmlPathDocument:** Provides a read-only cache for XML document processing by using XSLT. This class is optimizied for XSLT processing and does not check for the conformity of W3C DOM. For this reason, you can create an instance of this class to process an XML document faster. To create an instance of the XPathDocument class, you need to include the System.Xml.XPath namespace in the application. The Syntax to declare an object of this class is as follows:

  ```
  XmlPathDocument doc=new XmlPathDocument("XML1.xml");
  ```

- **XmlNodeReader:** Provides forward-only access to the data represented by the XmlDocument or XmlDataDocument class. If you want to create an instance of the XmlNodeReader class, you need to include the System.Xml namespace. The syntax to declare an object of this class is as follows:

  ```
  XmlDocument doc=new XmlPathDocument();

  XmlNodeReader reader=new XmlNodeReader(doc);
  ```

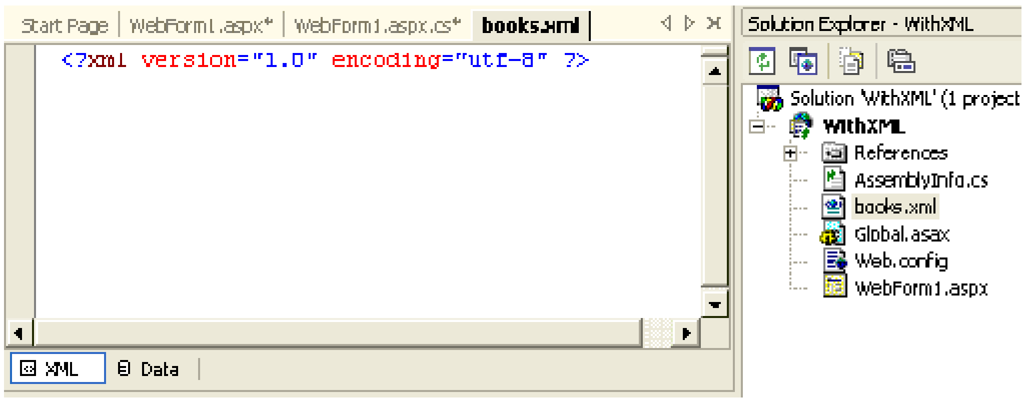
- **XslTransform:** Provides support for a XSLT 1.0 style sheet syntax that enables you to transform an XML document by using XSL style sheets. If you want to create an instance of the XslTransform class, you need to include the System.Xml.Xsl namespace in the application. The syntax to declare an object of this class is as follows:

  ```
  Xsltransform xslt = new XslTransform ();
  ```

1.2 XML Designer

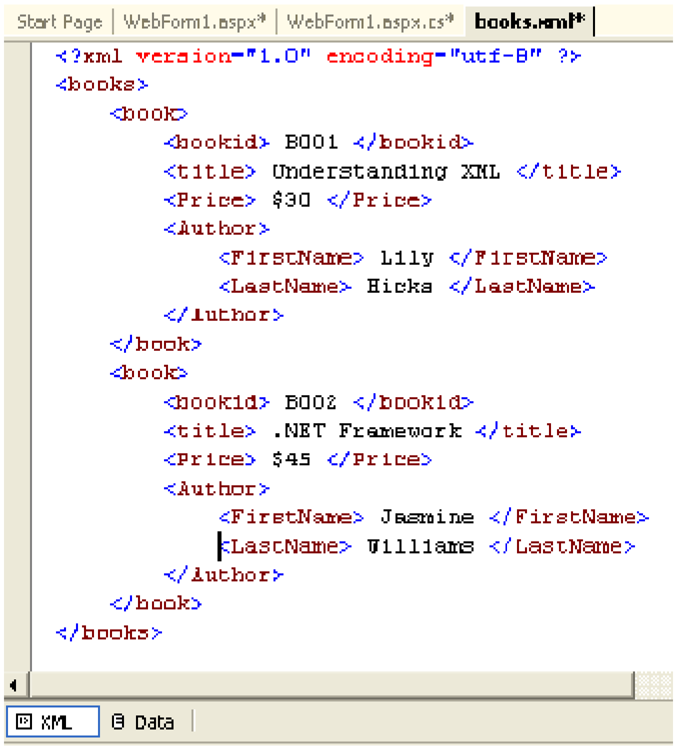
Visual Studio .NET provides the XML designer that you can use to create and edit XML documents. For example, if you need to create an XML document that contains the details of books available in an online bookstore, you need to perform the following steps by using the XML designer of Visual Studio .NET:

1. Create a new ASP.NET Web application.
2. Select the Add New Item option
3. Select XML File as the template from the right pane. Specify the name as "books.xml" and click open.
4. The XML designer is displayed. The XML designer has automatically generated a line that notifies the browser that the document being processed is an XML document, as displayed in the figure:



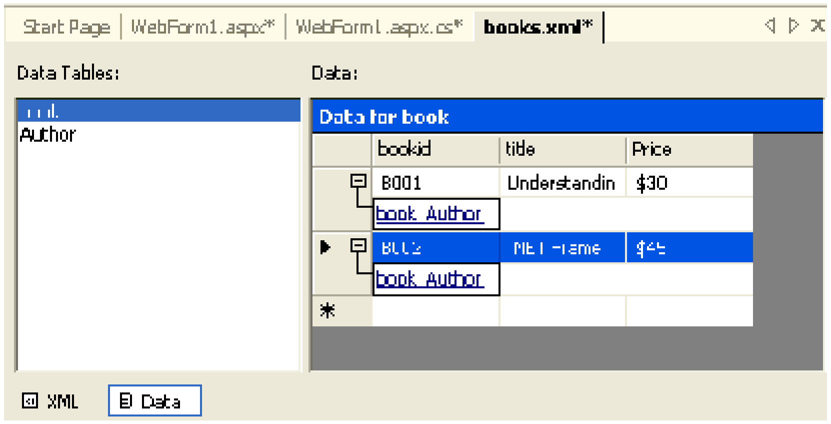
```
<?xml version="1.0" encoding="utf-8" ?>
```

5. At the bottom of the designer window, there are two tabs, ***XML*** and ***Data***. In the XML tab enter the following lines of code after the first line in the XML designer:



```
<?xml version="1.0" encoding="utf-8" ?>
<books>
    <book>
        <bookid> B001 </bookid>
        <title> Understanding XML </title>
        <Price> $30 </Price>
        <Author>
            <FirstName> Lily </FirstName>
            <LastName> Hicks </LastName>
        </Author>
    </book>
    <book>
        <bookid> B002 </bookid>
        <title> .NET Framework </title>
        <Price> $45 </Price>
        <Author>
            <FirstName> Jasmine </FirstName>
            <LastName> Williams </LastName>
        </Author>
    </book>
</books>
```

The ***Data*** view displays the XML data represented by the XML document. When you switch to the Data view, the data appears, as displayed in the following figure:


Start Page | WebForm1.aspx* | WebForm1.aspx.cs* | books.xml*

Data Tables:

book
Author

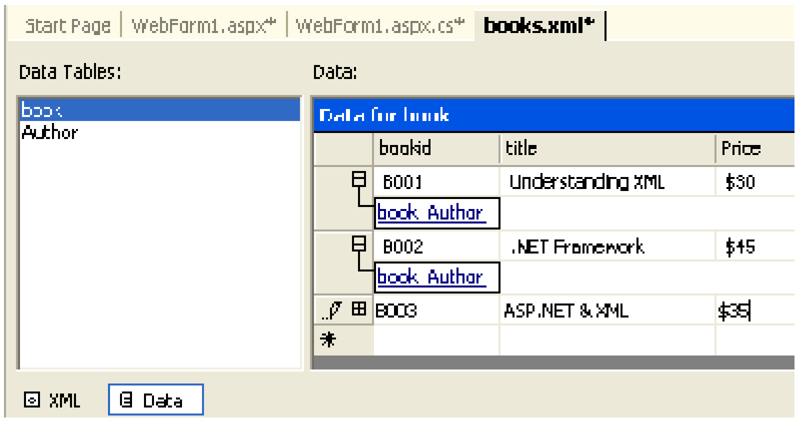
Data:

Data for book

	bookid	title	Price
⊟	B001	Understandin	$30
	book_Author		
▶ ⊟	B002	.NET Frame	$45
	book_Author		
*			

XML | Data

In addition to just viewing data in the Data view, you can also add data directly to an existing XML document. For this, just click on the new row below the existing data and enter your values, and shown in the figure:


Start Page | WebForm1.aspx* | WebForm1.aspx.cs* | books.xml*

Data Tables:

book
Author

Data:

Data for book

	bookid	title	Price
⊟	B001	Understanding XML	$30
	book_Author		
⊟	B002	.NET Framework	$45
	book_Author		
✎ ⊞	B003	ASP.NET & XML	$35
*			

XML | Data

1.3 XML Web Server Control

An XML Web Server control is used to display the contents of an XML document without formatting or using XSL Transformations. You can optionally specify a XSLT style sheet that formats the XML document before it is displayed in an XML server control. The XML Web Server control belongs to the System.Web.UI.WebControls namespace. You can add an XML Web Server control to a Web form by dragging the control from the Web forms tab of the toolbox.

The XML Web Server control has the following properties:

- ***DocumentSource***: Allows you to specify the URL or the path of the XML document to be displayed in the Web form.
- ***TransformSource:*** Allows you to specify the URL of the XSLT file, which transforms the XML document into the required format before it is displayed in the Web form.
- ***Document:*** Allows you to specify a reference to an object of the XMLDocument class. This property is available only at runtime.

- ***Transform:*** Allows you to specify a reference to an object of the XMLTransform class. This property is available only at runtime.

1.4 XML Document Object Model Support

When you want to access and display XML data in Web Applications, you use the XML Web server control and set its properties at design time. In certain situation, you may need to display the XML data based on specific conditions. In such cases, you will have to access the XML data programmatically.

An XML document consists of elements and attributes. For this reason, you can access XML data programmatically. Note that XML DOM allows you to access and manipulate the elements and attributes present in an XML document programmatically.

To implement XML DOM, the .NET framework provides a set of additional classes included in the System.Xml namespace. These classes enable you to access the XML data programmatically. Some of the classes in the System.Xml namespace are as follows:

- ***XmlDocumentType***: Represents the DTD used by an XML document.
- ***XmlElement:*** Represents one element from an XML document.
- ***XmlAttribute:*** Represents one attribute of an element.

2. XML Basics

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.The design goals of XML focus on simplicity,generality,and usability across the Internet.It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structuressuch as those used in web services.

1. XML stands for eXtensible Markup Language
2. XML is a markup language like HTML
3. XML is designed to store and transport data
4. XML is designed to be self-descriptive

Differences between XML and HTML
XML and HTML were designed with different goals:

- XML is designed to carry data emphasizing on what type of data it is.
- HTML is designed to display data emphasizing on how data looks
- XML tags are not predefined like HTML tags.
- HTML is a markup language whereas XML provides a framework for defining markup languages.
- HTML is about displaying data,hence it is static whereas XML is about carrying information,which makes it dynamic.

3. XML Attributes

The XML attribute is a part of an XML element. The addition of attribute in XML element gives more precise properties of the element i.e, it enhances the properties of the XML element.

Syntax:

```
<element_name attribute1 attribute2 ... > Contents...
</element_name>
```

In the above syntax element_name is the name of an element which can be any name. The attribute1, attribute2, … is XML attribute having unique attribute name. Then in the content section, any message can be written and at the end, the element name is ended.

Below some examples are given which illustrate the above syntax:

Example 1:

<text category = "message">Hello Geeks</text>

In the above example, XML element is text, the **category** is the attribute name and **message** is the attribute value, Attribute name and its value always appear in pair. The attribute name is used without any quotation but attribute value is used in single (‘ ‘) or double quotation (” ”).

Example 2:

<text category = "message" purpose ="Greet">Hello Geeks</text>

In the above example, two attribute is used with different name. So, in a single element multiple attribute is used having unique attribute name.

But if we use two distinct element then we can use the attribute having the same attribute name. This can be understood with the help of below example:

Example 3:

<text category = "message" >Hello Geeks.</text>

<text category = "message">How are you.</text>

3.1 Attribute Types:

There are three types of attributes discribed below:

- **String types Attribute:** This type of attribute takes any string literal as a value.
- **Enumerated Type Attribute:** This attribute is further distributed in two types-
    - **Notation Type:** This attribute is used to declares that an element will be referenced to a notation which is declared somewhere else in the XML document.

    - **Enumeration:** This attribute is used to specify a particular list of values which match with attribute values.

- **Tokenized Type Attribute:** This attribute is further distributed in many types:

    - **ID:** This attribute is used to identify the element.

    - **IDREF:** This attribute is used to refer an ID which has already been named for another element.

    - **IDREFS:** This attribute is used to refer all IDs of an element.

    - **ENTITY:** This attribute is used to indicate the attribute which will represent an external entity used in the document.

    - **ENTITIES:** This attribute is used to indicate the attribute which will represent external entities used in the document.

3.3 Rules for creating an attribute:

There are some rules that should be followed while creating an attribute:

- An attribute should not repeat itself in a single start or empty-element tag.

- An attribute should be declared using the attribute-list declaration in the DTD (Document Type Definition).

- An attribute element is used without any quotation and the attribute value is used in a single (‘ ‘) or double quotation (” “).

- An attribute name and its value should always appear in pair.

- An attribute value should not contain direct or indirect entity references to external entities.

4. Fundamental XML Classes

The eXtensive Markup Language has a very important role in web applications. .Net has given enough space for XML classes in its framework. The main classes working with XML data are as follows: -

XmlTextReader

The **XmlTextReader** class is an implementation of **XmlReader**.

- It provides a fast, performant parser.

- It enforces the rules that XML must be well formed.

- It is neither a validating nor a non-validating parser since it does not have DTD or schema information.

- It can read text in blocks, or read characters from a stream.

The **XmlTextReader** performs the following functions:

- Enforces the rules that XML must be well-formed.
- Checks that the DTD is well-formed. However, does not use the DTD for validation, expanding entity references, or adding default attributes.
- Validating is not done against DTDs or Schemas.
- Checks the correct formation of any DOCTYPE nodes.
- Checks the correct formation of the entities. For node types of **EntityReference**, a single empty **EntityReference** node is returned. An empty **EntityReference** node is one in which its **Value** property is a **string. Empty**. This is because there is no DTD or schema to expand the entity reference. The **XmlTextReader** does ensure that the whole DTD is well-formed, including the **EntityReference** nodes.
- Provides a performant XML parser, because the **XmlTextReader** does not havethe overhead involved with validation checking.

XMLTextWriter

Represents a writer that provides a fast, non-cached, forward the only way of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

XMLDocument

This class implements the W3C Document Object Model (DOM) Level 1 Core and the Core DOM Level 2. The DOM is an in-memory (cache) tree representation of an XML document that enables the navigation and editing of this document.

XmlDataDocument

This class Allows structured data to be stored, retrieved, and manipulated through a relational DataSet.

XmlNodeReader

The **XmlNodeReader** has the ability to read an XML DOM sub tree. This class does not support DTD or schema validation.

Document Navigator

This class enables to navigate an XML document represented by XmlDocument class.

5. XML Validations:

Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is deal in two ways by the XML parser. They are −

- Well-formed XML document
- Valid XML document

Well-formed XML Document

An XML document is said to be **well-formed** if it adheres to the following rules −

- Non DTD XML files must use the predefined character entities for **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)**.
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self ending tag.(<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- **amp(&)**, **apos(single quote)**, **gt(>)**, **lt(<)**, **quot(double quote)** entities other than these must be declared.

Example

Following is an example of a well-formed XML document −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address
[
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>

<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

The above example is said to be well-formed as −

- It defines the type of document. Here, the document type is **element** type.
- It includes a root element named as **address**.
- Each of the child elements among name, company and phone is enclosed in its self explanatory tag.
- Order of the tags is maintained.

5.1 Valid XML Document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document.

XML - DTDs

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

Syntax

Basic syntax of a DTD is as follows −

```
<!DOCTYPE element DTD identifier
[
   declaration1
   declaration2
   ........
]>
```

In the above syntax,

- The **DTD** starts with <!DOCTYPE delimiter.
- An **element** tells the parser to parse the document from the specified root element.
- **DTD identifier** is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called **External Subset.**
- **The square brackets [ ]** enclose an optional list of entity declarations called *Internal Subset*.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to **yes**. This means, the declaration works independent of an external source.

Syntax

Following is the syntax of internal DTD −

```
<!DOCTYPE root-element [element-declarations]>
```

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
<!DOCTYPE address [
   <!ELEMENT address (name,company,phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT company (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>

<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

Let us go through the above code −

Start Declaration − Begin the XML declaration with the following statement.

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

DTD − Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE −

```
<!DOCTYPE address [
```

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body − The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone_no (#PCDATA)>
```

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration − Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (**]**>). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) − it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD −

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage −

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
   <name>Tanmay Patil</name>
   <company>TutorialsPoint</company>
   <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file **address.dtd** is as shown −

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either **system identifiers** or **public identifiers**.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

<!DOCTYPE name SYSTEM "address.dtd" [...]>

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and is written as follows −

<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called **Formal Public Identifiers, or FPIs**.

5.2 XML - Schemas

XML Schema is commonly known as **XML Schema Definition (XSD)**. It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows –

Example

The following example shows how to use schema –

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema">
   <xs:element name = "contact">
      <xs:complexType>
         <xs:sequence>
            <xs:element name = "name" type = "xs:string" />
            <xs:element name = "company" type = "xs:string" />
            <xs:element name = "phone" type = "xs:int" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Elements

As we saw in the XML - Elements chapter, elements are the building blocks of XML document. An element can be defined within an XSD as follows –

```
<xs:element name = "x" type = "y"/>
```

Definition Types

You can define XML schema elements in the following ways –

Simple Type

Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example –

```
<xs:element name = "phone_number" type = "xs:int" />
```

Complex Type

A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example –

```
<xs:element name = "Address">
   <xs:complexType>
      <xs:sequence>
```

```
         <xs:element name = "name" type = "xs:string" />
         <xs:element name = "company" type = "xs:string" />
         <xs:element name = "phone" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other **<xs:element>** definitions, that allows to build a simple hierarchy of elements in the XML document.

Global Types

With the global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as follows −

```
<xs:element name = "AddressType">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "name" type = "xs:string" />
         <xs:element name = "company" type = "xs:string" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Now let us use this type in our example as follows −

```
<xs:element name = "Address1">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "address" type = "AddressType" />
         <xs:element name = "phone1" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>

<xs:element name = "Address2">
   <xs:complexType>
      <xs:sequence>
         <xs:element name = "address" type = "AddressType" />
         <xs:element name = "phone2" type = "xs:int" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below −

```
<xs:attribute name = "x" type = "y"/>
```

6. XML in ADO .Net

There are two approaches to work with XML and ADO. First, you can use ADO.NET to access XML documents. Second, you can use XML and ADO.NET to access XML. Additionally, you can access a relational database using ADO.NET and XML.NET.

Reading XML using Data Set

In ADO.NET, you can access the data using the DataSet class. The DataSet class implements methods and properties to work with XML documents. The following sections discuss methods that read XML data.

The Read xml Method

ReadXml is an overloaded method; you can use it to read a data stream, TextReader, XmlReader, or an XML file and to store into a DataSet object, which can later be used to display the data in a tabular format. The ReadXml method has eight overloaded forms. It can read a text, string, stream, TextReader, XmlReader, and their combination formats. In the following example, create a new DataSet object.

In the following example, create a new DataSet object and call the DataSet. ReadXml method to load the books.xml file in a DataSet object:

```
//Create a DataSet object

      DataSet ds = new DataSet();

      // Fill with the data

      ds.ReadXml("books.xml");
```

Once you've a DataSet object, you know how powerful it is. Make sure you provide the correct path of books.xml.

Note: Make sure you add a reference to System.Data and the System.Data.Common namespace before using DataSet and other common data components.

The ReadXmlSchema method

The ReadXMLSchema method reads an XML schema in a DataSet object. It has four overloaded forms. You can use a Text Reader, string, stream, and XmlReader. The

following example shows how to use a file as direct input and call the ReadXmlSchema method to read the file:

```
DataSet ds = new DataSet();
ds.ReadSchema(@"c:\books.xml");
```

The following example reads the file XmlReader and uses XmlTextReader as the input of ReadXmlSchema:

```
//Create a dataset object

      DataSet ds = new DataSet("New DataSet");

      // Read xsl in an XmlTextReader

      XmlTextReader myXmlReader = new XmlTextReader(@"c:\books.Xml");

      // Call Read xml schema

      ds.ReadXmlSchema(myXmlReader);

      myXmlReader.Close();
```

The Writexml Method

The WriteXml method writes the current data (the schema and data) of a DataSet object to an XML file. This is overloaded method. By using this method, you can write data to a file, stream, TextWriter, or XmlWriter. This example creates a DataSet, fills the data for the DataSet, and writes the data to an XML file.

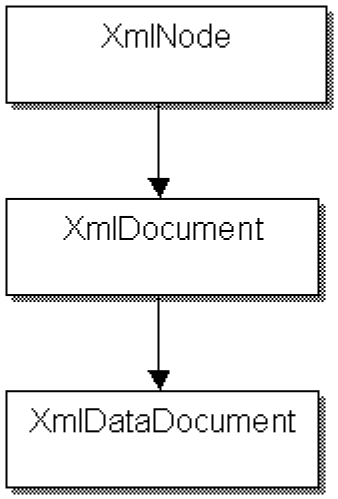
The Write xml schema method

This method writes DataSet structure to an XML schema. WriteXmlSchema has four overloaded methods. You can write the data to a stream, text, TextWriter, or Xmlwriter.

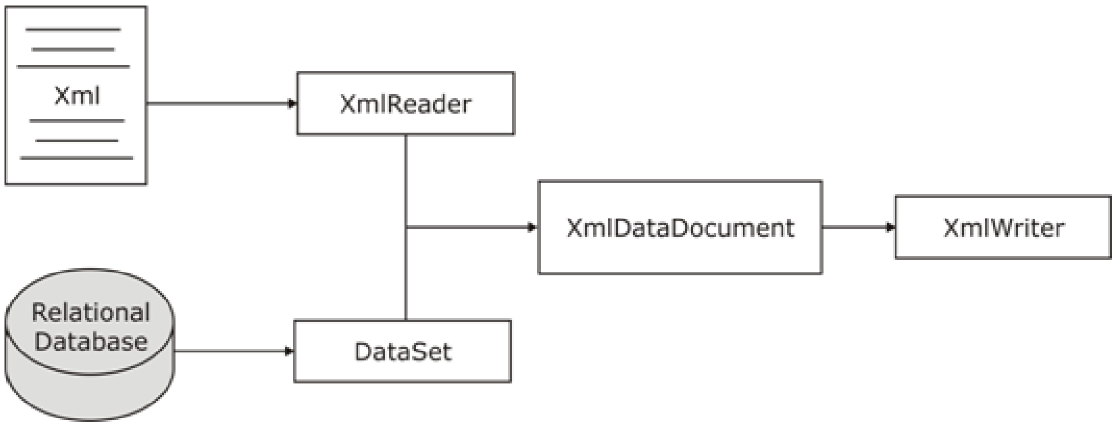
7. Xml Data Document

the XmlDocument class provides DOM tree structure of XML documents. The XmlDataDocument class comes from XmlDocument, which is comes from XmlNode.

XML Data document Hierarchy



Besides overriding the methods of XmlNode and XmlDocument, XmlDataDocument also implements its own methods. The XmlDataDocument class lets you lead relational data using the DataSet object as well as XML documents using the Load and LoadXml methods. As figure given below indicates, you can use a DataSet to load relational data to an XmlDataDocument object and use the Load or LoadXml methods to read an XML document. Below Figure shows a relationship between a Reader, Writer, DataSet, and XmlDataDocument.



The XmlDataDocument class extends the functionality of XmlDocument and synchronizes it with DataSet. As you know a DataSet is a powerful object in ADO.NET. As above given figure shows, you can take data from two different sources. First, you can load data from an XML document with the help of XmlReader, and second, you can load data from relational data sources with the help of database provides and DataSet. The neat thing is the data synchronization between these two objects. That means if you update data in a DataSet object, you see results in the XmlDataDocument object and vice versa. For example, if you add a record to a DataSet object, the action will add one node to the XmlDataDocument object representing the newly added record.

Once the data is loaded, you're allowed to use any operations that you were able to use on XmlDocument objects. You can also use XmlReader and XmlWriter objects to read and write the data.

The xmlData Documet class has property called DataSet. It returns the attached DataSet object with XmlDataDocument. The DataSet property provides you a relational representation of an XML document. Once you've a DataSet object, you can do anything with it such as attaching to a DataGrid.

You Can use all XML read and write methods of the DataSet object through the DataSet property such as ReadXml, ReadXmlSchema, WriteXml, and WriteXml schema.

7.1 Loading Data using Load and LoadXml from the XmlDataDocument

You can use either the Load method or the LoadXml method to load an XML document. The Load method takes a parameter of a filename string, a TextReader, or an XmlReader. Similarly, you can use the LoadXml method. This method passes an XML file name to load the XML file for example:

```
XmlDataDocument doc = new XmlDataDocument();

doc.Load("c:\\Books.xml");
```

Or you can load an XML fragment, as in the following example:

```
 XmlDataDocument doc = new XmlDataDocument();

 doc.LoadsXml("<Record> write something </Record>");
```

7.2 Loading Data Using a DataSet

A DataSet object has methods to read XML documents. These methods are ReadXmlSchema and LoadXml. You use the Load or LoadXml methods to load an XML document the same way you did directly from the XMLDataDocument. Again the Load method takes a parameter of a filename string, TextReader, or XmlReader. Similarly, use the LoadXml method to pass an XML filename through the dataset. For example:

```
 XmlDataDocument doc = new XmlDataDocument();

 doc.DataSet.ReadXmlSchema("test. Xsd")
```

Or

```
 doc.DataSet.ReadXml("<Record> write something </Record>");
```

7.3 Displaying XML Data In a data Set Format

As mentioned previously, you can get DataSet object from an XmlDataDocument object by using its DataSet property. OK, now it's time to see how to do that. The next sample will show you how easy is to display an XML document data in a DataSet format.

To read XML document in a dataset, first you read to document. You can read a document using the ReadXml method of the DataSet object. The DataSet property of XmlDataDocument represents the dataset of XmlDataDocument. After reading a document in a dataset, you can create data views from the dataset, or you can also use a DataSet'sDefaultViewManager property to bind to data-bound controls, as you can see in the following code:

```
XmlDataDocument xmlDatadoc = new XmlDataDocument();

xmlDatadoc.DataSet.ReadXml("c:\\ xmlDataDoc.xml");

 dataGrid1.DataSource = xmlDatadoc.DataSet.DefaultViewManager;
```

As you can see from below given code, I created a new dataset, Books, fill from the books.xml and bind to a DataGrid control using its DataSource property. To make to do, you need to create a Windows application and drag a DataGrid control to the form. After doing that, you need to write the code on the Form1 constructor or Form load event.

Code:

```
public Form1( ) {

        // Initialize Component and other code here

        // Create an XmlDataDocument object and read an XML

        XmlDataDocument xmlDatadoc = new XmlDataDocument();

        xmlDatadoc.DataSet.ReadXml("C:\\books.xml");

        // Create a DataSet object and fill with the dataset

        // of XmlDataDocument

        DataSet ds = new DataSet("Books DataSet");

        ds = xmlDatadoc.DataSet;

        // Attach dataset view to the Data Grid control

        dataGrid1.DataSource = ds.DefaultViewManager;
```

7.4 Saving Data from a DataSet to XML

You can save a DataSet data as an XML document using the Save method of XmlDataDocument. Actually, XmlDataDocument comes from XmlDocument., and the XmlDocument class defines the Save method. I've already discussed that you can use Save method to save your data in a string, stream, TextWriter, and XmlWriter.

First, you create a DataSet object and fill it using a DataAdapter. The following example reads the Customers table from the Northwind Access database and fills data from the read to the DataSet:

```
string SQLStmt = "SELECT * FROM Customers";

string ConnectionString =

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C: \\ Northwind.mdb";
// Create data adapter

OleDbDataAdapter da = new OleDbDataAdapter(SQLStmt,
ConnectionString);

 // create a new dataset object and fill using data adapter's fill method

DataSet ds = new DataSet();

da.Fill(ds);
```

Now, you create an instance of XmlDataDocument with the DataSet as an argument and call the Save method to save the data as an XML document:

```
XmlDataDocument doc = new XmlDataDocument(ds);

doc.Save("C:\\XmlDataDoc.xml");
```

Code given below shows a complete program. You create an XmlDataDocument object with dataset and call the save method to save the dataset data in an XML file.

Code: Saving the dataset data to an XML document

```
using System;

using System.Data;
using System.Data.OleDb;

using System.Xml;



namespace DataDocsampB2

{

        class Class1

        {

                static void Main(string[] args)

                {// create SQL Query

                string SQLStmt = "SELECT * FROM Customers";

                // Connection string

                string ConnectionString =

                "Provider = Microsoft.Jet.OLEDB.4.0;Data Source = C:\\ Northwind.mdb";

                // Create data adapter

                OleDbDataAdapter da = new OleDbDataAdapter(SQLStmt,
        ConnectionString);

                // create a new dataset object and fill using data adapter's fill method

                DataSet doc = new DataSet();

                // Now use SxlDataDocument's Save method to save data as an XML file
        XmlDataDocument doc = new XmlDataDocument(ds);

                doc.Save("C:\\XmlDataDoc.xml");

                }

        }

}
```

8. **What is a web service?**

A "web service is the communication platform between two different or same platform applications that allows to use their web method." In the preceding definition you observed that there are two main points in the definition of web service; they are different or same platform application and the second is web method.

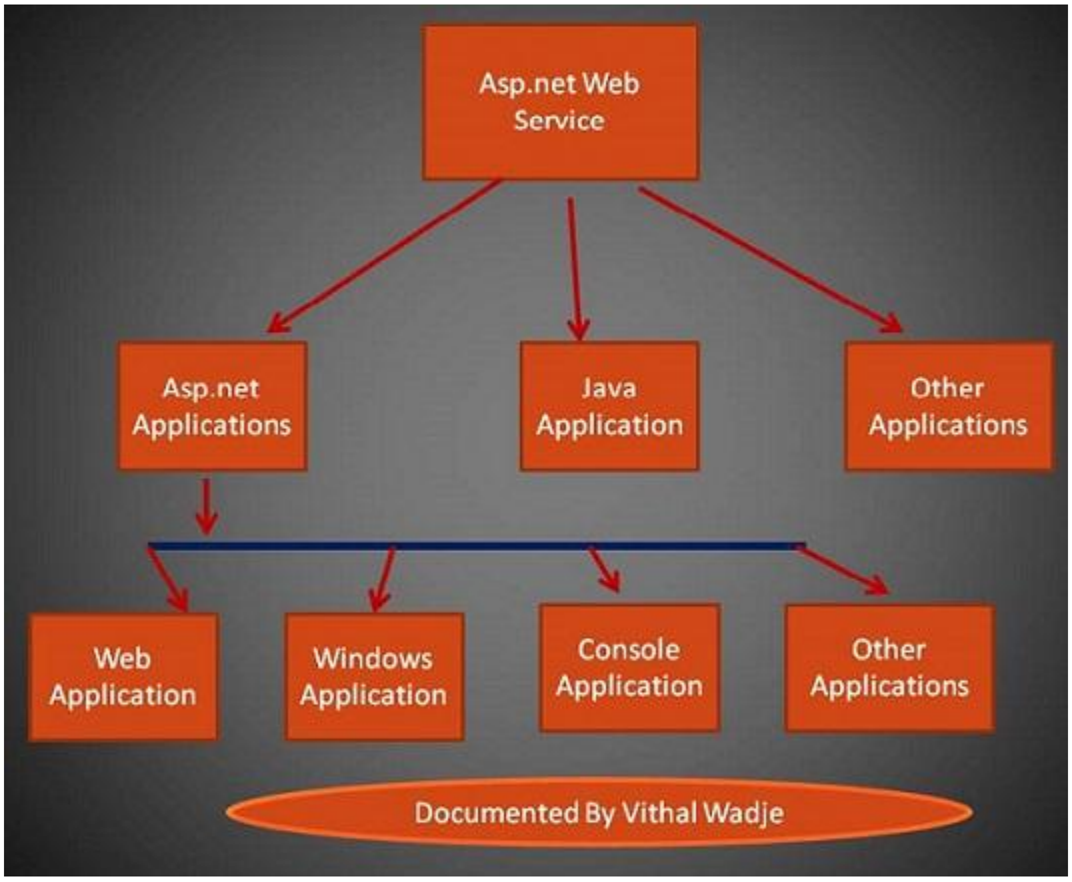
What does different or same application and platform mean?

It means that we can create a web service in any language, such as Java or other languages and that the language web service can be used in a .Net based application and also a .Net web service or in another application to exchange the information.

What does* web method *mean

The method in web services always start with [webMethod] attributes, it means that it is a web method that is accessible anywhere, the same as my web application.

All above given definitions can be explained by following diagram

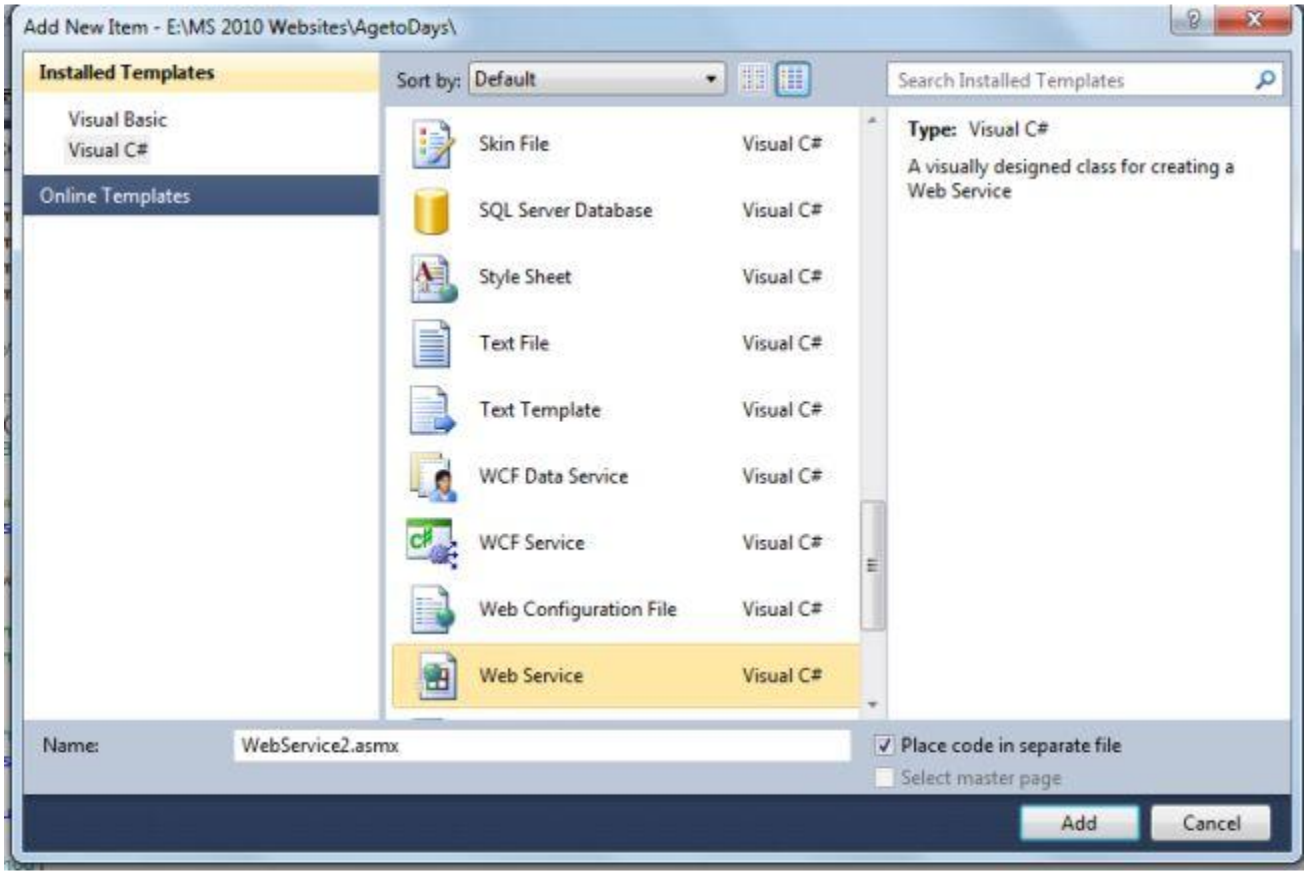


In the above diagram, we have shown, how the Asp.net Web Service is used in different types of applications means we are trying to explain that we can create a web service in any language, such as Java or other languages and that the language web service can be used in a .Net based application as well as java or other applications and you can also use .Net based web application in Java applications means web Services don't have a any platform restrictions.

8.1 Using Web Service

Add a web service using a template

1. "Start" - "All Programs" - "Microsoft Visual Studio 2010"
2. "File" - "New Project" - "C#" - "Empty Web Application" (to avoid adding a master page)
3. Provide the web site a name such as "agetodays" or another as you wish and specify the location
4. Then right-click on Solution Explorer - "Add New Item" - you see the web service templates



Add New Item - E:\MS 2010 Websites\AgetoDays\

Installed Templates: Visual Basic, Visual C#

Online Templates

Sort by: Default

Skin File	Visual C#
SQL Server Database	Visual C#
Style Sheet	Visual C#
Text File	Visual C#
Text Template	Visual C#
WCF Data Service	Visual C#
WCF Service	Visual C#
Web Configuration File	Visual C#
Web Service	Visual C#

Search Installed Templates

Type: Visual C#

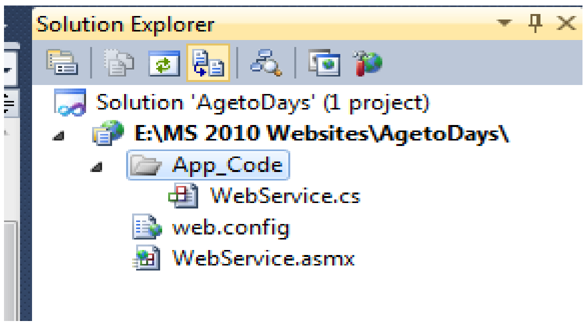
A visually designed class for creating a Web Service

Name: WebService2.asmx

[✓] Place code in separate file

[ ] Select master page

Select Web Service Template and click on add button. then after that the Solution Explorer look like as follows.



Solution 'AgetoDays' (1 project)
E:\MS 2010 Websites\AgetoDays\
App_Code
WebService.cs
web.config
WebService.asmx

Then open the Webservice.cs class and write the following method followed by [webMethod] attribute as in.

```
[WebMethod]
public int converttodaysweb(int day, int month, int year) {

    DateTime dt = new DateTime(year, month, day);
    int datetodays = DateTime.Now.Subtract(dt).Days;
    return datetodays;
}
```

In the code above we have declared a one integer method named converttodaysweb with the three parameters day, month and year for accepting day, month and year from the user.

Then after that we have created an object of date time and ed the those variables that we get from the users. We declared another variable in the method that is age today to store the number of days remaining from the user's input date to the current date and finally we return that variable..

The webservice.cs file will then look as in the following

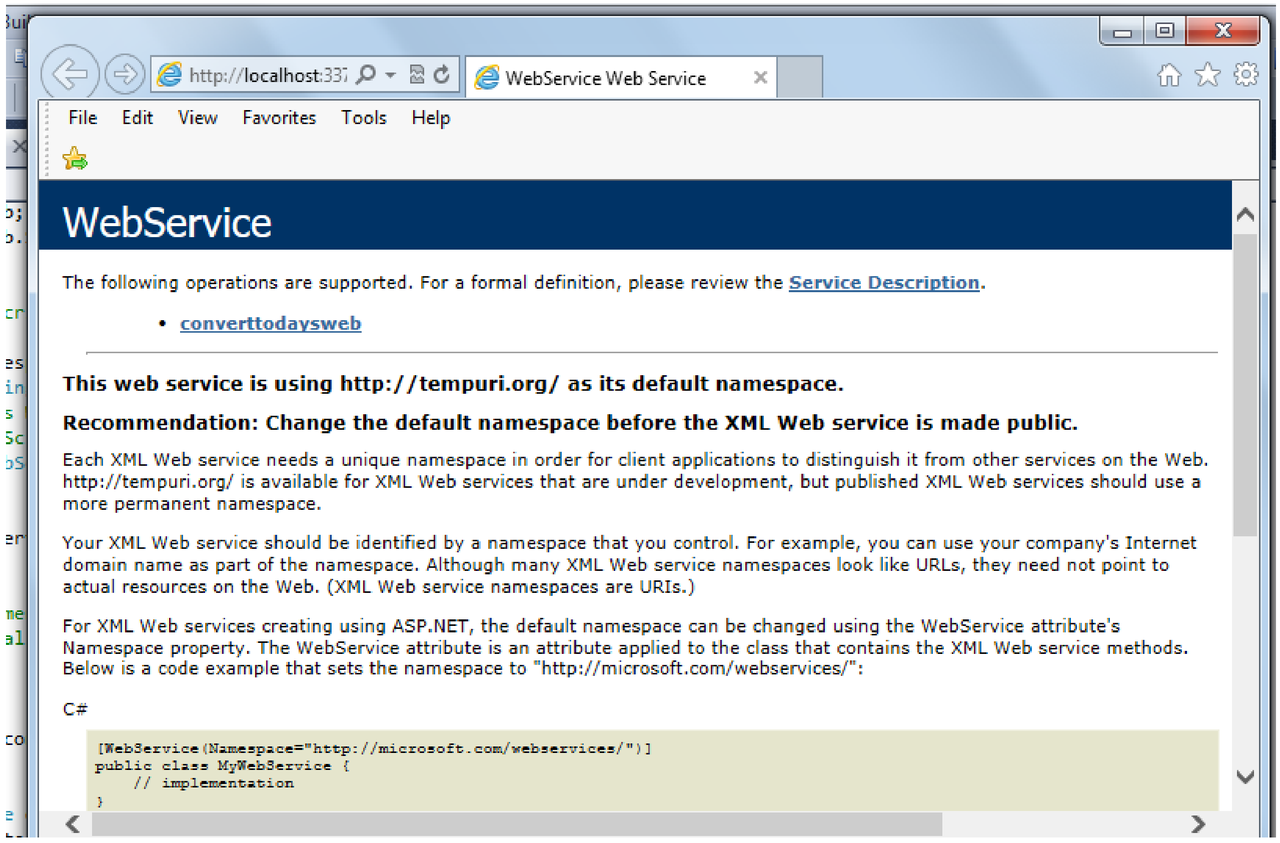
```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.Services;
///<summary>
/// Summary description for UtilityWebService
///</summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)
]
     // To allow this Web Service to be called from script,
 using ASP.NET AJAX, uncomment the following line.
     // [System.Web.Script.Services.ScriptService]
     public class WebService: System.Web.Services.WebServic
e
     {
         public WebService()
         {
             //Uncomment the following line if using designed components
             //InitializeComponent();
         }
         [WebMethod]
         public int converttodaysweb(int day, int month, int year)
         {
             DateTime dt = new DateTime(year, month, day);

             int datetodays = DateTime.Now.Subtract(dt).Days;
             return datetodays;

         }

     }
```

Now run the application that look like as follows.



WebService

The following operations are supported. For a formal definition, please review the Service Description.

- converttodaysweb

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

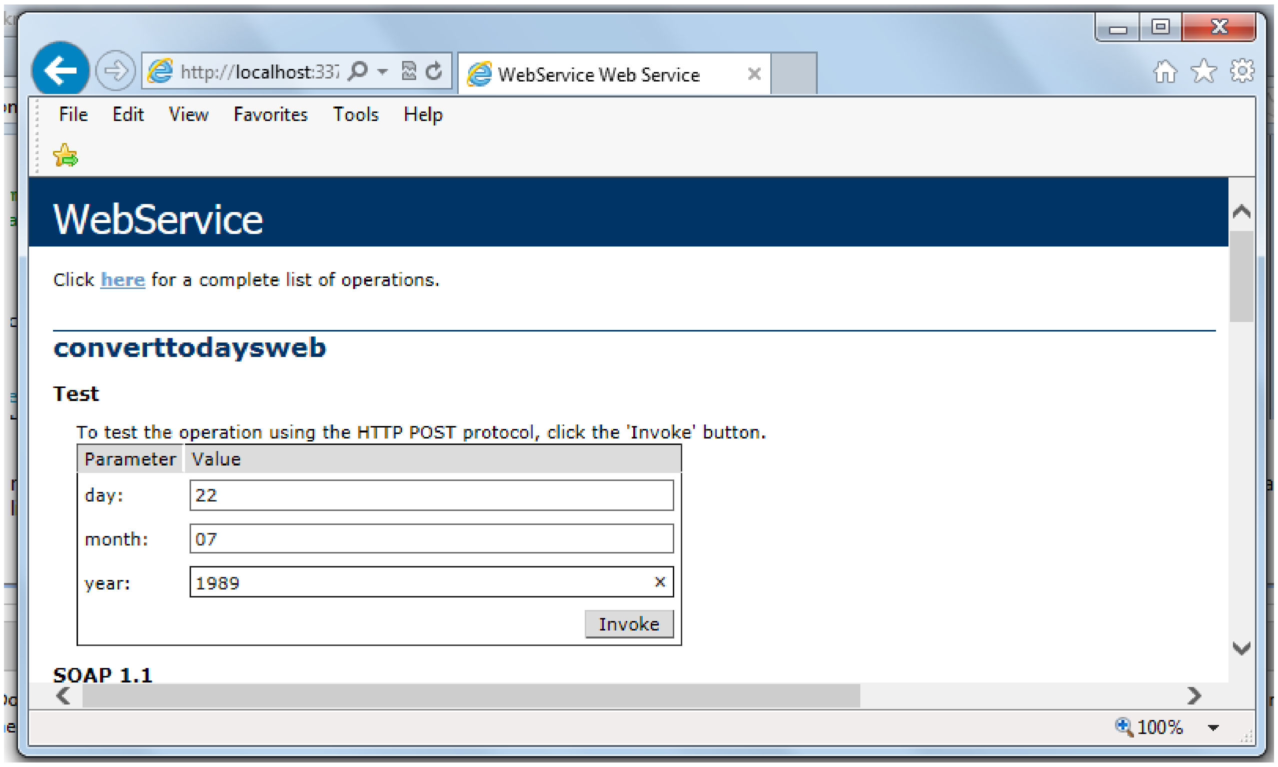
Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Now in the above we see our method that we are created in the webservice.cs file, so click on that method and provide input values and click on the "invoke" link as in.



WebService

Click here for a complete list of operations.

converttodaysweb

Test

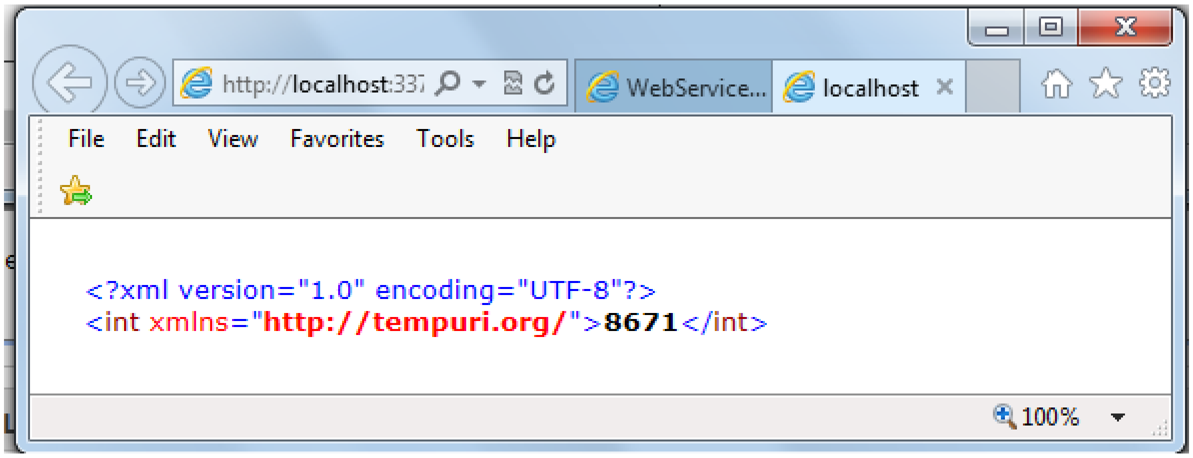
To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
day:	22
month:	07
year:	1989

Invoke

SOAP 1.1

The output will be as follows



```
<?xml version="1.0" encoding="UTF-8"?>
<int xmlns="http://tempuri.org/">8671</int>
```

in the screen above you see that the output is 8671, that is the days from the input date.

9. State Management

State management means to preserve state of a control, web page, object/data, and user in the application explicitly because all ASP.NET web applications are stateless, i.e., by default, for each page posted to the server, the state of controls is lost. Nowadays all web apps demand a high level of state management from control to application level.

Types of state management

There are two types of state management techniques: client side and server side.

Client side

1. Hidden Field
2. View State
3. Cookies
4. Control State
5. Query Strings

Server side

1. Session
2. Application

9.1 Levels of state management

1. Control level: In ASP.NET, by default controls provide state management automatically.
2. Variable or object level: In ASP.NET, member variables at page level are stateless and thus we need to maintain state explicitly.
3. Single or multiple page level: State management at single as well as multiple page level i.e., managing state between page requests.
4. User level: State should be preserved as long as a user is running the application.
5. Application level: State available for complete application irrespective of the user, i.e., should be available to all users.
6. Application to application level: State management between or among two or more applications.

9.2 Client side methods

1. Hidden field

Hidden field is a control provided by ASP.NET which is used to store small amounts of data on the client. It store one value for the variable and it is a preferable way when a variable's value is changed frequently. Hidden field control is not rendered to the client (browser) and it is invisible on the browser. A hidden field travels with every request like a standard control’s value.
Let us see with a simple example how to use a hidden field. These examples increase a value by 1 on every "No Action Button" click. The source of the hidden field control is.

```
<asp:HiddenField ID="HiddenField1" runat="server"  />
```
In the code-behind page:

```
protected void Page_Load(object sender, EventArgs e)
{
  if (HiddenField1.Value != null)
  {
   int val= Convert.ToInt32(HiddenField1.Value) + 1;
   HiddenField1.Value = val.ToString();
   Label1.Text = val.ToString();
  }
}
protected void Button1_Click(object sender, EventArgs e)
{
 //this is No Action Button Click
}
```

2. View state

View state is another client side state management mechanism provided by ASP.NET to store user's data, i.e., sometimes the user needs to preserve data temporarily after a post back, then the view state is the preferred way for doing it. It stores data in the generated HTML using hidden field not on the server. View State provides page level state management i.e., as long as the user is on the current page, state is available and the user redirects to the next page and the current page state is lost. View State can store any type of data because it is object type but it is preferable not to store a complex type of data due to the need for serialization and deserilization on each post back. View state is enabled by default for all server side controls of ASP.NET with a property EnableviewState set to true.

Let us see how ViewState is used with the help of the following example. In the example we try to save the number of postbacks on button click.

```
protected void Page_Load(object sender, EventArgs e)
{
   if (IsPostBack)
   {
      if (ViewState["count"] != null)
      {
         int ViewstateVal = Convert.ToInt32(ViewState["count"]) + 1;
         Label1.Text = ViewstateVal.ToString();
         ViewState["count"]=ViewstateVal.ToString();
      }
      else
      {
         ViewState["count"] = "1";
      }
   }
}
protected void Button1_Click(object sender, EventArgs e)
{
```

Label1.Text=ViewState["count"].ToString();}

3. Cookies

Cookie is a small text file which is created by the client's browser and also stored on the client hard disk by the browser. It does not use server memory. Generally a cookie is used to identify users.

A cookie is a small file that stores user information. Whenever a user makes a request for a page the first time, the server creates a cookie and sends it to the client along with the requested page and the client browser receives that cookie and stores it on the client machine either permanently or temporarily (persistent or non persistence). The next time the user makes a request for the same site, either the same or another page, the browser checks the existence of the cookie for that site in the folder. If the cookie exists it sends a request with the same cookie, else that request is treated as a new request.

Types of Cookies

1. Persistence Cookie: Cookies which you can set an expiry date time are called persistence cookies. Persistence cookies are permanently stored till the time you set.

Let us see how to create persistence cookies. There are two ways, the first one is:

```
Response.Cookies["nameWithPCookies"].Value = "This is A Persistance Cookie";
Response.Cookies["nameWithPCookies"].Expires = DateTime.Now.AddSeconds(10);
```

And the second one is:

```
HttpCookie aCookieValPer = new HttpCookie("Persistance");
aCookieValPer.Value = "This is A Persistance Cookie";
aCookieValPer.Expires = DateTime.Now.AddSeconds(10);
Response.Cookies.Add(aCookieValPer);
```

2. Non-Persistence Cookie:

Non persistence cookies are not permanently stored on the user client hard disk folder. It maintains user information as long as the user accesses the same browser. When user closes the browser the cookie will be discarded. Non Persistence cookies are useful for public computers.

Let us see how to create a non persistence cookies. There are two ways, the first one is:

```
Response.Cookies["nameWithNPCookies"].Value = "This is A Non Persistance
Cookie";
```

And the second way is:

```
HttpCookie aCookieValNonPer = new HttpCookie("NonPersistance");
aCookieValNonPer.Value = "This is A Non Persistance Cookie;
```

```
Response.Cookies.Add(aCookieValNonPer);how to create cookie :
```

How to read a cookie:

```
if (Request.Cookies["NonPersistance"] != null)
Label2.Text = Request.Cookies["NonPersistance"].Value;
```

Limitation of cookies:

The number of cookies allowed is limited and varies according to the browser. Most browsers allow 20 cookies per server in a client's hard disk folder and the size of a cookie is not more than 4096 bytes or 4 KB of data that also includes name and value data.

4. Control State

Control State is another client side state management technique. Whenever we develop a custom control and want to preserve some information, we can use view state but suppose view state is disabled explicitly by the user, the control will not work as expected. For expected results for the control we have to use Control State property. Control state is separate from view state.

How to use control state property:

Control state implementation is simple. First override the OnInit() method of the control and add a call for the Page.RegisterRequiresControlState() method with the instance of the control to register. Then override LoadControlState and SaveControlState in order to save the required state information.

9.3 Server side

1. Session

Session management is a very strong technique to maintain state. Generally session is used to store user's information and/or uniquely identify a user (or say browser). The server maintains the state of user information by using a session ID. When users makes a request without a session ID, ASP.NET creates a session ID and sends it with every request and response to the same user.

How to get and set value in Session:

```
Session["Count"] = Convert.ToInt32(Session["Count"]) + 1;//Set Value to The
Session
Label2.Text = Session["Count"].ToString(); //Get Value from the Sesion
```

Session Events in ASP.NET

To manage a session, ASP.NET provides two events:

session_start and session_end that is written in a special file called *Global.asax* in the root directory of the project.

Session_Start:

The Session_start event is raised every time a new user makes a request without a session ID, i.e., new browser accesses the application, then a session_start event raised. Let's see the *Global.asax* file.

```
void Session_Start(object sender, EventArgs e)
{
   Session["Count"] = 0;  // Code that runs when a new session is started
}
```

Session_End: The Session_End event is raised when session ends either because of a time out expiry or explicitly by using Session.Abandon(). The Session_End event is raised only in the case of In proc mode not in the state server and SQL Server modes.

There are four session storage mechanisms provided by ASP.NET:

- In Proc mode
- State Server mode
- SQL Server mode
- Custom mode

In Process mode:

In proc mode is the default mode provided by ASP.NET. In this mode, session values are stored in the web server's memory (in IIS). If there are more than one IIS servers then session values are stored in each server separately on which request has been made. Since the session values are stored in server, whenever server is restarted the session values will be lost.

```
<configuration>
 <sessionstate mode="InProc" cookieless="false" timeout="10"

    stateConnectionString="tcpip=127.0.0.1:80808"

    sqlConnectionString="Data Source=.\SqlDataSource;User
ID=userid;Password=password"/>
</configuration>
```

In State Server mode:

This mode could store session in the web server but out of the application pool. But usually if this mode is used there will be a separate server for storing sessions, i.e., stateServer. The benefit is that when IIS restarts the session is available. It stores session in a separate Windows service. For State server session mode, we have to configure it explicitly in the web config file and start the aspnet_state service.

```
<configuration><sessionstate mode="stateserver" cookieless="false"

   timeout="10"  stateConnectionString="tcpip=127.0.0.1:42424"

   sqlConnectionString="Data Source=.\SqlDataSource;User
ID=userid;Password=password"/> </configuration>
```

In SQL Server mode:

Session is stored in a SQL Server database. This kind of session mode is also separate from IIS, i.e., session is available even after restarting the IIS server. This mode is highly secure and reliable but also has a disadvantage that there is overhead from serialization and deserialization of session data. This mode should be used when reliability is more important than performance.

```
<configuration>
    <sessionstate mode="sqlserver" cookieless="false" timeout="10"

       stateConnectionString="tcpip=127.0.0.1:4  2424"

       sqlConnectionString="Data Source=.\SqlDataSource;User
ID=userid;Password=password"/>
</configuration>
```

Custom Session mode:

Generally we should prefer in proc state server mode or SQL Server mode but if you need to store session data using other than these techniques then ASP.NET provides a custom session mode. This way we have to maintain everything customized even generating session ID, data store, and also security.

Attributes	**Description**
Cookieless true/false	Indicates that the session is used with or without cookie. cookieless set to true indicates sessions without cookies is used and cookieless set to false indicates sessions with cookies is used. cookieless set to false is the default set.
timeout	Indicates the session will abound if it is idle before session is abounded explicitly (the default time is 20 min).

StateConnectionString	Indicates the session state is stored on the remote computer (server). This attribute is required when session mode is StateServer
SqlConnectionString	Indicates the session state is stored in the database. This attribute is required when session mode is SqlServer.

2. Application

Application state is a server side state management technique. The date stored in application state is common for all users of that particular ASP.NET application and can be accessed anywhere in the application. It is also called application level state management. Data stored in the application should be of small size.

How to get and set a value in the **application object**:

```
Application["Count"] = Convert.ToInt32(Application["Count"]) + 1; //Set Value
to The Application Object
Label1.Text = Application["Count"].ToString(); //Get Value from the
Application Object
```

Application events in ASP.NET

There are three types of events in ASP.NET. Application event is written in a special file called *Global.asax*. This file is not created by default, it is created explicitly by the developer in the root directory. An application can create more than one *Global.asax* file but only the root one is read by ASP.NET.

Application_start: The Application_Start event is raised when an app domain starts. When the first request is raised to an application then the Application_Start event is raised. Let's see the *Global.asax* file.

```
void Application_Start(object sender, EventArgs e)
{
    Application["Count"] = 0;
}
```

Application_Error:

It is raised when an unhandled exception occurs, and we can manage the exception in this event.

Application_End:

The Application_End event is raised just before an application domain ends because of any reason, may IIS server restarting or making some changes in an application cycle.

10. SOAP (Simple Object Access Protocol)

SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.

SOAP is known as the Simple Object Access Protocol, but in later times was just shortened to SOAP v1.2. SOAP is a protocol or in other words is a definition of how web services talk to each other or talk to client applications that invoke them.

SOAP was developed as an intermediate language so that applications built on various programming languages could talk easily to each other and avoid the extreme development effort.

10.1 SOAP Introduction

In today's world, there is huge number of applications which are built on different programming languages. For example, there could be a web application designed in Java, another in .Net and another in PHP.

Exchanging data between applications is crucial in today's networked world. But data exchange between these heterogeneous applications would be complex. So will be the complexity of the code to accomplish this data exchange.

One of the methods used to combat this complexity is to use XML (Extensible Markup Language) as the intermediate language for exchanging data between applications.

Every programming language can understand the XML markup language. Hence, XML was used as the underlying medium for data exchange.

But there are no standard specifications on use of XML across all programming languages for data exchange. That is where SOAP comes in.

SOAP was designed to work with XML over HTTP and have some sort of specification which could be used across all applications. We will look into further details on the SOAP protocol in the subsequent chapters.

10.2 Advantages of SOAP

SOAP is the protocol used for data interchange between applications. Below are some of the reasons as to why SOAP is used.

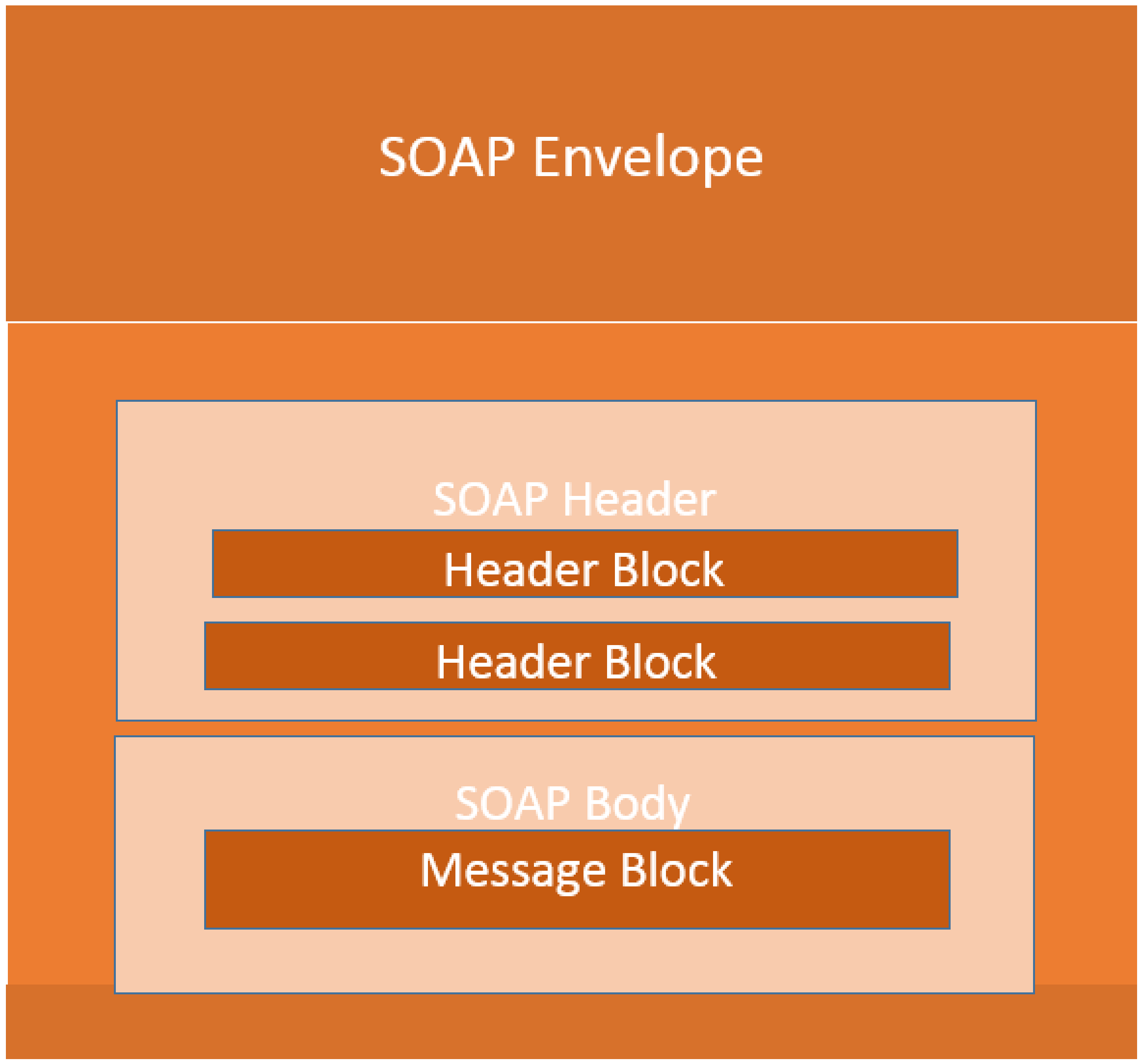
- When developing Web services, you need to have some of language which can be used for web services to talk with client applications. SOAP is the perfect medium which was developed in order to achieve this purpose. This protocol is also recommended by the W3C consortium which is the governing body for all web standards.

- SOAP is a light-weight protocol that is used for data interchange between applications. Note the keyword '**light**.' Since SOAP is based on the XML language, which itself is a light weight data interchange language, hence SOAP as a protocol that also falls in the same category.
- SOAP is designed to be platform independent and is also designed to be operating system independent. So the SOAP protocol can work any programming language based applications on both Windows and Linux platform.
- It works on the HTTP protocol –SOAP works on the HTTP protocol, which is the default protocol used by all web applications. Hence, there is no sort of customization which is required to run the web services built on the SOAP protocol to work on the World Wide Web.

10.3 SOAP Building blocks

The SOAP specification defines something known as a "**SOAP message**" which is what is sent to the web service and the client application.

The diagram below shows the various building blocks of a SOAP Message.



The SOAP message is nothing but a mere XML document which has the below components.

- An Envelope element that identifies the XML document as a SOAP message – This is the containing part of the SOAP message and is used to encapsulate all the details in the SOAP message. This is the root element in the SOAP message.
- A Header element that contains header information – The header element can contain information such as authentication credentials which can be used by the calling application. It can also contain the definition of complex types which could be used in the SOAP message. By default, the SOAP message can contain parameters which could be of simple types such as strings and numbers, but can also be a complex object type.

11. Web Service Description Language

Web Services Description Language (WSDL) is a format for describing a Web Services interface. It is a way to describe services and how they should be bound to specific network addresses. WSDL has three parts:

- Definitions
- Operations
- Service bindings

Definitions are generally expressed in XML and include both data type definitions and message definitions that use the data type definitions. These definitions are usually based upon some agreed upon XML vocabulary. This agreement could be within an organization or between organizations. Vocabularies within an organization could be designed specifically for that organization. They may or may not be based on some industry-wide vocabulary. If data type and message definitions need to be used between organizations, then most likely an industry-wide vocabulary will be used.

XML, however, is not necessary required for definitions. The OMG Interface Definition Language (IDL), for example, could be used instead of XML. If a different definitional format were used, senders and receivers would need to agree on the format as well as the vocabulary. Nevertheless, over time, XML-based vocabularies and messages are likely to dominate. XML Namespaces are used to ensure uniqueness of the XML element names in the definitions, operations, and service bindings.

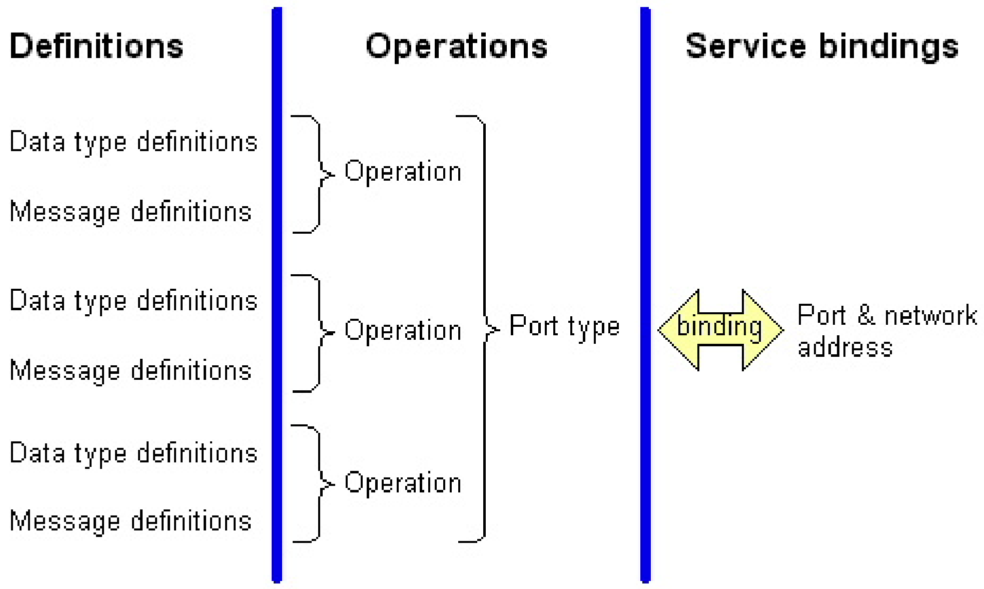
Operations describe actions for the messages supported by a Web service. There are four types of operations:

- One-way: Messages sent without a reply required
- Request/response: The sender sends a message and the received sends a reply.
- Solicit response: A request for a response. (The specific definition for this action is pending.)
- Notification: Messages sent to multiple receivers. (The specific definition for this action is pending.)

Operations are grouped into port types. Port types define a set of operations supported by the Web service.

Service bindings connect port types to a port. A port is defined by associating a network address with a port type. A collection of ports defines a service. This binding is commonly created using SOAP, but other forms may be used. These other forms could include CORBA Internet Inter-ORB Protocol (IIOP), DCOM, .NET, Java Message Service (JMS), or WebSphere MQ to name a few.

The following figure shows the relationship of the basic parts of WSDL:



References:

- http://www.beansoftware.com/ASP.NET-Tutorials/Using-XML.aspx
- https://www.geeksforgeeks.org/xml-basics/
- https://www.tutorialspoint.com/xml
- https://www.c-sharpcorner.com/UploadFile/mahesh/ado-net-and-xml/
- https://www.c-sharpcorner.com/UploadFile/0c1bb2/converting-from-to-to-current-date-into-days-using-Asp-Net/
- https://www.codeproject.com/Articles/492397/State-Management-in-ASP-NET-Introduction
- https://www.guru99.com/soap-simple-object-access-protocol.html