WEB SERVICES

web application components

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Web services are open standard (XML, SOAP, HTTP, etc.) based web applications that interact with other web applications for the purpose of exchanging data

Web services can convert your existing applications into web applications.

In this tutorial, you will learn what exactly web services are and why and how to use them.

Audience

This tutorial will be useful for all those readers inclined to learn the basics of web services and implement them in practice.

Prerequisites

This is an elementary tutorial that introduces the concepts of web services. It does not require the readers to have a prior knowledge of any technology in particular, however it would certainly make you comfortable if you have a basic understanding of XML, HTTP, TCP/IP concepts.

Copyright & Disclaimer

Table of Contents

1. WHAT ARE WEB SERVICES?

Different books and different organizations provide different definitions to Web Services. Some of them are listed here.

- A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.
- Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
- Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

To summarize, a complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism

Components of Web Services

The basic web services platform is XML + HTTP. All the standard web services work using the following components:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

All these components have been discussed in the Web Services Architecture chapter.

How Does a Web Service Work?

A web service enables communication among various applications by using open standards such as HTML, XML, WSDL, and SOAP. A web service takes the help of:

- XML to tag the data
- SOAP to transfer a message
- WSDL to describe the availability of service.

You can build a Java-based web service on Solaris that is accessible from your Visual Basic program that runs on Windows.

You can also use C# to build new web services on Windows that can be invoked from your web application that is based on JavaServer Pages (JSP) and runs on Linux.

Example

Consider a simple account-management and order processing system. The accounting personnel use a client application built with Visual Basic or JSP to create new accounts and enter new customer orders.

The processing logic for this system is written in Java and resides on a Solaris machine, which also interacts with a database to store information.

The steps to perform this operation are as follows:

- The client program bundles the account registration information into a SOAP message.
- This SOAP message is sent to the web service as the body of an HTTP POST request.
- The web service unpacks the SOAP request and converts it into a command that the application can understand.
- The application processes the information as required and responds with a new unique account number for that customer.
- Next, the web service packages the response into another SOAP message, which it sends back to the client program in response to its HTTP request.
- The client program unpacks the SOAP message to obtain the results of the account registration process.

2. WHY WEB SERVICES?

Here are the benefits of using Web Services:

Exposing the Existing Function on the Network

A web service is a unit of managed code that can be remotely invoked using HTTP. That is, it can be activated using HTTP requests. Web services allow you to expose the functionality of your existing code over the network. Once it is exposed on the network, other applications can use the functionality of your program.

Interoperability

Web services allow various applications to talk to each other and share data and services among themselves. Other applications can also use the web services. For example, a VB or .NET application can talk to Java web services and vice versa. Web services are used to make the application platform and technology independent.

Standardized Protocol

Web services use standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) use well-defined protocols in the web services protocol stack. This standardization of protocol stack gives the business many advantages such as a wide range of choices, reduction in the cost due to competition, and increase in the quality.

Low Cost Communication

Web services use SOAP over HTTP protocol, so you can use your existing low-cost internet for implementing web services. This solution is much less costly compared to proprietary solutions like EDI/B2B. Besides SOAP over HTTP, web services can also be implemented on other reliable transport mechanisms like FTP.

3. CHARACTERISTICS

Web services have the following special behavioral characteristics:

XML-Based

Web services use XML at data representation and data transportation layers. Using XML eliminates any networking, operating system, or platform binding. Web services based applications are highly interoperable at their core level.

Loosely Coupled

A consumer of a web service is not tied to that web service directly. The web service interface can change over time without compromising the client's ability to interact with the service. A tightly coupled system implies that the client and server logic are closely tied to one another, implying that if one interface changes, the other must be updated. Adopting a loosely coupled architecture tends to make software systems more manageable and allows simpler integration between different systems.

Coarse-Grained

Object-oriented technologies such as Java expose their services through individual methods. An individual method is too fine an operation to provide any useful capability at a corporate level. Building a Java program from scratch requires the creation of several fine-grained methods that are then composed into a coarse-grained service that is consumed by either a client or another service.

Businesses and the interfaces that they expose should be coarse-grained. Web services technology provides a natural way of defining coarse-grained services that access the right amount of business logic.

Ability to be Synchronous or Asynchronous

Synchronicity refers to the binding of the client to the execution of the service. In synchronous invocations, the client blocks and waits for the service to complete its operation before continuing. Asynchronous operations allow a client to invoke a service and then execute other functions.

Asynchronous clients retrieve their result at a later point in time, while synchronous clients receive their result when the service has completed. Asynchronous capability is a key factor in enabling loosely coupled systems.

Supports Remote Procedure Calls (RPCs)

Web services allow clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol. Remote procedures expose input and output parameters that a web service must support.

Component development through Enterprise JavaBeans (EJBs) and .NET Components has increasingly become a part of architectures and enterprise deployments over the past couple of years. Both technologies are distributed and accessible through a variety of RPC mechanisms.

A web service supports RPC by providing services of its own, equivalent to those of a traditional component, or by translating incoming invocations into an invocation of an EJB or a .NET component.

Supports Document Exchange

One of the key advantages of XML is its generic way of representing not only data, but also complex documents. These documents can be as simple as representing a current address, or they can be as complex as representing an entire book or Request for Quotation (RFQ). Web services support the transparent exchange of documents to facilitate business integration.

4. ARCHITECTURE

There are two ways to view the web service architecture:

- The first is to examine the individual roles of each web service actor.
- The second is to examine the emerging web service protocol stack.

Web Service Roles

There are three major roles within the web service architecture:

Service Provider

This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service Requestor

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service Registry

This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.

Web Service Protocol Stack

A second option for viewing the web service architecture is to examine the emerging web service protocol stack. The stack is still evolving, but currently has four main layers.

Service Transport

This layer is responsible for transporting messages between applications. Currently, this layer includes Hyper Text Transport Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and newer protocols such as Blocks Extensible Exchange Protocol (BEEP).

XML Messaging

This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

Service Description

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

Service Discovery

This layer is responsible for centralizing services into a common registry and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

As web services evolve, additional layers may be added and additional technologies may be added to each layer.

The next chapter explains the components of web services.

Few Words about Service Transport

The bottom of the web service protocol stack is service transport. This layer is responsible for actually transporting XML messages between two computers.

Hyper Text Transfer Protocol (HTTP)

Currently, HTTP is the most popular option for service transport. HTTP is simple, stable, and widely deployed. Furthermore, most firewalls allow HTTP traffic. This allows XMLRPC or SOAP messages to masquerade as HTTP messages. This is good if you want to integrate remote applications, but it does raise a number of security concerns.

Blocks Extensible Exchange Protocol (BEEP)

This is a promising alternative to HTTP.. BEEP is a new Internet Engineering Task Force (IETF) framework for building new protocols. BEEP is layered directly on TCP and includes a number of built-in features, including an initial handshake protocol, authentication, security, and error handling. Using BEEP, one can create new protocols for a variety of applications, including instant messaging, file transfer, content syndication, and network management.

SOAP is not tied to any specific transport protocol. In fact, you can use SOAP via HTTP, SMTP, or FTP. One promising idea is therefore to use SOAP over BEEP.

5. COMPONENTS

Over the past few years, three primary technologies have emerged as worldwide standards that make up the core of today's web services technology. These technologies are discussed below.

XML-RPC

This is the simplest XML-based protocol for exchanging information between computers.

- XML-RPC is a simple protocol that uses XML messages to perform RPCs.
- Requests are encoded in XML and sent via HTTP POST.
- XML responses are embedded in the body of the HTTP response.
- XML-RPC is platform-independent.
- XML-RPC allows diverse applications to communicate.
- A Java client can speak XML-RPC to a Perl server.
- XML-RPC is the easiest way to get started with web services.

To learn more about XML-RPC, visit our XML-RPC Tutorial.

SOAP

SOAP is an XML-based protocol for exchanging information between computers.

- SOAP is a communication protocol.
- SOAP is for communication between applications.
- SOAP is a format for sending messages.
- SOAP is designed to communicate via Internet.
- SOAP is platform independent.
- SOAP is language independent.
- SOAP is simple and extensible.
- SOAP allows you to get around firewalls.
- SOAP will be developed as a W3C standard.

To learn more about SOAP, visit our SOAP Tutorial.

WSDL

WSDL is an XML-based language for describing web services and how to access them.

- WSDL stands for Web Services Description Language.
- WSDL was developed jointly by Microsoft and IBM.
- WSDL is an XML based protocol for information exchange in decentralized and distributed environments.
- WSDL is the standard format for describing a web service.
- WSDL definition describes how to access a web service and what operations it will perform.
- WSDL is a language for describing how to interface with XML-based services.
- WSDL is an integral part of UDDI, an XML-based worldwide business registry.
- WSDL is the language that UDDI uses.
- WSDL is pronounced as 'wiz-dull' and spelled out as 'W-S-D-L'

To learn more about WSDL, visit our WSDL Tutorial.

UDDI

UDDI is an XML-based standard for describing, publishing, and finding web services.

- UDDI stands for Universal Description, Discovery, and Integration.
- UDDI is a specification for a distributed registry of web services.
- UDDI is platform independent, open framework.
- UDDI can communicate via SOAP, CORBA, and Java RMI Protocol.
- UDDI uses WSDL to describe interfaces to web services.
- UDDI is seen with SOAP and WSDL as one of the three foundation standards of web services.
- UDDI is an open industry initiative enabling businesses to discover each other and define how they interact over the Internet.

To learn more about UDDI, visit our UDDI Tutorial.

6. EXAMPLE

Based on the web service architecture, we create the following two components as a part of web services implementation:

Service Provider or Publisher

This is the provider of the web service. The service provider implements the service and makes it available on the Internet or intranet.

We will write and publish a simple web service using .NET SDK.

Service Requestor or Consumer

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

We will also write two web service requestors: one web-based consumer (ASP.NET application) and another Windows application-based consumer.

Given below is our first web service example which works as a service provider and exposes two methods (add and SayHello) as the web services to be used by applications. This is a standard template for a web service. .NET web services use the .asmx extension. Note that a method exposed as a web service has the WebMethod attribute. Save this file as FirstService.asmx in the IIS virtual directory (as explained in configuring IIS; for example, c:\MyWebSerces).

```
FirstService.asmx
  <%@ WebService language="C" class="FirstService" %>


  using System;
  using System.Web.Services;
  using System.Xml.Serialization;


  [WebService(Namespace="http://localhost/MyWebServices/")]
  public class FirstService : WebService
  {
      [WebMethod]
      public int Add(int a, int b)
      {
          return a + b;
```

```
        }


        [WebMethod]
        public String SayHello()
        {
            return "Hello World";
        }
    }
```

To test a web service, it must be published. A web service can be published either on an intranet or the Internet. We will publish this web service on IIS running on a local machine. Let us start with configuring the IIS.

- Open Start -> Settings -> Control Panel -> Administrative tools -> Internet Services Manager.
- Expand and right-click on the default website; select New -> Virtual Directory. The Virtual Directory Creation Wizard opens. Click Next.
- The "Virtual Directory Alias" screen opens. Type the virtual directory name. For example, MyWebServices. Click Next.
- The "Web Site Content Directory" screen opens.
- Enter the directory path name for the virtual directory. For example, c:\MyWebServices. Click Next.
- The "Access Permission" screen opens. Change the settings as per your requirements. Let us keep the default settings for this exercise.
- Click the Next button. It completes the IIS configuration.
- Click Finish to complete the configuration.

To test whether the IIS has been configured properly, copy an HTML file (For example, x.html) in the virtual directory (C:\MyWebServices) created above. Now, open Internet Explorer and type http://localhost/MyWebServices/x.html. It should open the x.html file.

Note: If it does not work, try replacing the localhost with the IP address of your machine. If it still does not work, check whether IIS is running; you may need to reconfigure the IIS and the Virtual Directory.

To test this web service, copy FirstService.asmx in the IIS virtual directory created above (C:\MyWebServices). Open the web service in Internet Explorer (http://localhost/MyWebServices/FirstService.asmx). It should open your web service page. The page should have links to two methods exposed as web services by our application. Congratulations! You have written your first web service!

Testing the Web Service

As we have just seen, writing web services is easy in the .NET Framework. Writing web service consumers is also easy in the .NET framework; however, it is a bit more involved. As said earlier, we will write two types of service consumers, one web-based and another Windows application-based consumer. Let us write our first web service consumer.

Web-Based Service Consumer

Write a web-based consumer as given below. Call it WebApp.aspx. Note that it is an ASP.NET application. Save this in the virtual directory of the web service (c:\MyWebServices\WebApp.axpx).

This application has two text fields that are used to get numbers from the user to be added. It has one button, Execute, that when clicked gets the Add and SayHello web services.

```
WebApp.axpx
  <%@ Page Language="C#" %>
  <script runat="server">
  void runSrvice_Click(Object sender, EventArgs e)
  {
      FirstService mySvc = new FirstService();
      Label1.Text = mySvc.SayHello();
      Label2.Text = mySvc.Add(Int32.Parse(txtNum1.Text),
                    Int32.Parse(txtNum2.Text)).ToString();
  }
  </script>
  <html>
  <head>
  </head>
  <body>
  <form runat="server">
    <p>
        <em>First Number to Add </em>:
```

```
        <asp:TextBox id="txtNum1" runat="server"
             Width="43px">4</asp:TextBox>
    </p>
    <p>
        <em>Second Number To Add </em>:
        <asp:TextBox id="txtNum2" runat="server"
             Width="44px">5</asp:TextBox>
    </p>
    <p>
        <strong><u>Web Service Result -</u></strong>
    </p>
    <p>
        <em>Hello world Service</em> :
        <asp:Label id="Label1" runat="server"
             Font-Underline="True">Label</asp:Label>
    </p>


    <p>
        <em>Add Service</em> :
        & <asp:Label id="Label2" runat="server"
               Font-Underline="True">Label</asp:Label>
    </p>
    <p align="left">
        <asp:Button id="runSrvice" onclick="runSrvice_Click"
             runat="server" Text="Execute"></asp:Button>
    </p>
  </form>
  </body>
  </html>
```

After the consumer is created, we need to create a proxy for the web service to be consumed. This work is done automatically by Visual Studio .NET for us when referencing a web service that has been added. Here are the steps to be followed:

- Create a proxy for the Web Service to be consumed. The proxy is created using the WSDL utility supplied with the .NET SDK. This utility extracts information from the Web Service and creates a proxy. The proxy is valid only for a particular Web Service. If you need to consume other Web Services, you need to create a proxy for this service as well. Visual Studio .NET creates a proxy automatically for you when the Web Service reference is added. Create a proxy for the Web Service using the WSDL utility supplied with the .NET SDK. It will create FirstSevice.cs file in the current directory. We need to compile it to create FirstService.dll (proxy) for the Web Service.

```
c:> WSDL http://localhost/MyWebServices/


                FirstService.asmx?WSDL

c:> csc /t:library FirstService.cs
```

- Put the compiled proxy in the bin directory of the virtual directory of the Web Service (c:\MyWebServices\bin). Internet Information Services (IIS) looks for the proxy in this directory.

- Create the service consumer, in the same way we already did. Note that an object of the Web Service proxy is instantiated in the consumer. This proxy takes care of interacting with the service.

- Type the URL of the consumer in IE to test it (for example, http://localhost/MyWebServices/WebApp.aspx).

Windows Application-Based Web Service Consumer

Writing a Windows application-based web service consumer is the same as writing any other Windows application. You only need to create the proxy (which we have already done) and reference this proxy when compiling the application. Following is our Windows application that uses the web service. This application creates a web service object (of course, proxy) and calls the SayHello, and Add methods on it.

```
WinApp.cs
  using System;
  using System.IO;

  namespace SvcConsumer{
  class SvcEater
  {
      public static void Main(String[] args)
      {
          FirstService mySvc = new FirstService();

          Console.WriteLine("Calling Hello World Service: " +  mySvc.SayHello());
          Console.WriteLine("Calling Add(2, 3) Service: " +
                                    mySvc.Add(2, 3).ToString());
        }
    }
}
```

Compile it using c:>csc /r:FirstService.dll WinApp.cs. It will create WinApp.exe. Run it to test the application and the web service.

Now, the question arises: How can you be sure that this application is actually calling the web service?

It is simple to test. Stop your web server so that the web service cannot be contacted. Now, run the WinApp application. It will fire a runtime exception. Now, start the web server again. It should work.

7. SECURITY

Security is critical to web services. However, neither XML-RPC nor SOAP specifications make any explicit security or authentication requirements.

There are three specific security issues with web services:

- Confidentiality
- Authentication
- Network Security

Confidentiality

If a client sends an XML request to a server, can we ensure that the communication remains confidential?

Answer lies here:

- XML-RPC and SOAP run primarily on top of HTTP.
- HTTP has support for Secure Sockets Layer (SSL).
- Communication can be encrypted via SSL.
- SSL is a proven technology and widely deployed.

A single web service may consist of a chain of applications. For example, one large service might tie together the services of three other applications. In this case, SSL is not adequate; the messages need to be encrypted at each node along the service path, and each node represents a potential weak link in the chain. Currently, there is no agreed-upon solution to this issue, but one promising solution is the W3C XML Encryption Standard. This standard provides a framework for encrypting and decrypting entire XML documents or just portions of an XML document. You can check it at http://www.w3.org/Encryption.

Authentication

If a client connects to a web service, how do we identify the user? Is the user authorized to use the service?

The following options can be considered but there is no clear consensus on a strong authentication scheme.

- HTTP includes built-in support for Basic and Digest authentication, and services can therefore be protected in much the same manner as HTML documents are currently protected.

- SOAP Digital Signature (SOAP-DSIG) leverages public key cryptography to digitally sign SOAP messages. It enables the client or server to validate the identity of the other party. Check it at http://www.w3.org/TR/SOAP-dsig.

- The Organization for the Advancement of Structured Information Standards (OASIS) is working on the Security Assertion Markup Language (SAML).

Network Security

There is currently no easy answer to this problem, and it has been the subject of much debate. For now, if you are truly intent on filtering out SOAP or XML-RPC messages, one possibility is to filter out all HTTP POST requests that set their content type to text/xml.

Another alternative is to filter the SOAPAction HTTP header attribute. Firewall vendors are also currently developing tools explicitly designed to filter web service traffic.

8. STANDARDS

This chapter gives you an idea of all the latest standards related to web services.

Transports

BEEP, the Blocks Extensible Exchange Protocol (formerly referred to as BXXP), is a framework for building application protocols. It has been standardized by IETF and it does for Internet protocols what XML has done for data.

- Blocks Extensible Exchange Protocol (BEEP)

Messaging

These messaging standards and specifications are intended to give a framework for exchanging information in a decentralized, distributed environment.

- SOAP 1.1 (Note)
- SOAP 1.2 (Specification)
- Web Services Attachments Profile 1.0
- SOAP Message Transmission Optimization Mechanism

Description and Discovery

Web services are meaningful only if potential users may find information sufficient to permit their execution. The focus of these specifications and standards is the definition of a set of services supporting the description and discovery of businesses, organizations, and other web services providers; the web services they make available; and the technical interfaces which may be used to access those services.

- UDDI 3.0
- WSDL 1.1 (Note)
- WSDL 1.2 (Working draft)
- WSDL 2.0 (Working Group)

Security

Using these security specifications, applications can engage in secure communication designed to work with the general web services framework.

- Web Services Security 1.0
- Security Assertion Markup Language (SAML)

Management

Web services manageability is defined as a set of capabilities for discovering the existence, availability, health, performance, usage, as well as the control and configuration of a web service within the web services architecture. As web services become pervasive and critical to business operations, the task of managing and implementing them is imperative to the success of business operations.

- Web Services Distributed Management

9. SUMMARY

In this tutorial, you have learnt how to use web services. However, a web service also include components such as WSDL, UDDI, and SOAP that contribute to make it active. The next step is to learn WSDL, UDDI, and SOAP.

WSDL

WSDL is an XML-based language for describing web services and how to access them.

WSDL describes a web service, along with the message format and protocol details for the web service.

To learn more about WSDL, visit our WSDL Tutorial.

UDDI

UDDI is an XML-based standard for describing, publishing, and finding web services.

To learn more about UDDI, visit our UDDI Tutorial.

SOAP

SOAP is a simple XML-based protocol that allows applications to exchange information over HTTP.

To learn more about SOAP, visit our SOAP Tutorial.