**Distributed Object Protocols**

**SOAP**

Soap is the communications protocol for XML Web services. When SOAP is described as a communications protocol, most people think of DCOM or CORBA and start asking things like, "How does SOAP do object activation?" or "What naming service does SOAP use?" While a SOAP implementation will probably include these things, the SOAP standard doesn't specify them. SOAP is a specification that defines the XML format for messages—and that's about it for the required parts of the spec. If you have a well-formed XML fragment enclosed in a couple of SOAP elements, you have a SOAP message. Simple isn't it?

There are other parts of the SOAP specification that describe how to represent program data as XML and how to use SOAP to do Remote Procedure Calls. These optional parts of the specification are used to implement RPC-style applications where a SOAP message containing a callable function, and the parameters to pass to the function, is sent from the client, and the server returns a message with the results of the executed function. Most current implementations of SOAP support RPC applications because programmers who are used to doing COM or CORBA applications understand the RPC style. SOAP also supports document style applications where the SOAP message is just a wrapper around an XML document. Document-style SOAP applications are very flexible and many new XML Web services take advantage of this flexibility to build services that would be difficult to implement using RPC.

The last optional part of the SOAP specification defines what an HTTP message that contains a SOAP message looks like. This HTTP binding is important because HTTP is supported by almost all current OS's (and many not-so-current OS's). The HTTP binding is optional, but almost all SOAP implementations support it because it's the only standardized protocol for SOAP. For this reason, there's a common misconception that SOAP requires HTTP. Some implementations support MSMQ, MQ Series, SMTP, or TCP/IP transports, but almost all current XML Web services use HTTP because it is ubiquitous. Since HTTP is a core protocol of the Web, most organizations have a network infrastructure that supports HTTP and people who understand how to manage it already. The security, monitoring, and load-balancing infrastructure for HTTP are readily available today.

A major source of confusion when getting started with SOAP is the difference between the SOAP specification and the many implementations of the SOAP specification. Most people who use SOAP don't write SOAP messages directly but use a SOAP toolkit to create and parse the SOAP messages. These toolkits generally translate function calls from some kind of language to a SOAP message. For example, the Microsoft SOAP Toolkit 2.0 translates COM function calls to SOAP and the Apache Toolkit translates JAVA function calls to SOAP. The types of function calls and the datatypes of the parameters supported vary with each SOAP implementation so a function that works with one toolkit may not work with another. This isn't a limitation of SOAP but rather of the particular implementation you are using.

By far the most compelling feature of SOAP is that it has been implemented on many different hardware and software platforms. This means that SOAP can be used to link disparate systems within and without your organization. Many attempts have been made in the past to come up with a common communications protocol that could be used for systems integration, but none of them have had the widespread adoption that SOAP has. Why is this? Because SOAP is much smaller and simpler to implement than many of the previous protocols. DCE and CORBA for example took years to implement, so only a few implementations were ever released. SOAP, however, can use existing XML Parsers and HTTP libraries to do most of the hard work, so a SOAP implementation can be completed in a matter of months. This is why there are more than 70 SOAP implementations available. SOAP obviously doesn't do everything that DCE or CORBA do, but the lack of complexity in exchange for features is what makes SOAP so readily available.

The ubiquity of HTTP and the simplicity of SOAP make them an ideal basis for implementing XML Web services that can be called from almost any environment. For more information on SOAP, see the MSDN [SOAP](http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000523) home page.

**What About Security?**

One of the first questions newcomers to SOAP ask is how does SOAP deal with security. Early in its development, SOAP was seen as an HTTP-based protocol so the assumption was made that HTTP security would be adequate for SOAP. After all, there are thousands of Web applications running today using HTTP security so surely this is adequate for SOAP. For this reason, the current SOAP standard assumes security is a transport issue and is silent on security issues.

When SOAP expanded to become a more general-purpose protocol running on top of a number of transports, security became a bigger issue. For example, HTTP provides several ways to authenticate which user is making a SOAP call, but how does that identity get propagated when the message is routed from HTTP to an SMTP transport? SOAP was designed as a building-block protocol, so fortunately, there are already specifications in the works to build on SOAP to provide additional security features for Web services. The [WS-Security specification](http://msdn.microsoft.com/en-us/library/ms951257.aspx) defines a complete encryption system.

**WSDL**

WSDL (often pronounced whiz-dull) stands for Web Services Description Language. For our purposes, we can say that a WSDL file is an XML document that describes a set of SOAP messages and how the messages are exchanged. In other words, WSDL is to SOAP what IDL is to CORBA or COM. Since WSDL is XML, it is readable and editable but in most cases, it is generated and consumed by software.

To see the value of WSDL, imagine you want to start calling a SOAP method provided by one of your business partners. You could ask him for some sample SOAP messages and write your application to produce and consume messages that look like the samples, but this can be error-prone. For example, you might see a customer ID of 2837 and assume it's an integer when in fact it's a string. WSDL specifies what a request message must contain and what the response message will look like in unambiguous notation.

The notation that a WSDL file uses to describe message formats is based on the XML Schema standard which means it is both programming-language neutral and standards-based which makes it suitable for describing XML Web services interfaces that are accessible from a wide variety of platforms and programming languages. In addition to describing message contents, WSDL defines where the service is available and what communications protocol is used to talk to the service. This means that the WSDL file defines everything required to write a program to work with an XML Web service. There are several tools available to read a WSDL file and generate the code required to communicate with an XML Web service. Some of the most capable of these tools are in Microsoft Visual Studio® .NET.

**UDDI**

Universal Discovery Description and Integration is the yellow pages of Web services. As with traditional yellow pages, you can search for a company that offers the services you need, read about the service offered and contact someone for more information. You can, of course, offer a Web service without registering it in UDDI, just as you can open a business in your basement and rely on word-of-mouth advertising but if you want to reach a significant market, you need UDDI so your customers can find you.

A UDDI directory entry is an XML file that describes a business and the services it offers. There are three parts to an entry in the UDDI directory. The "white pages" describe the company offering the service: name, address, contacts, etc. The "yellow pages" include industrial categories based on standard taxonomies such as the North American Industry Classification System and the Standard Industrial Classification. The "green pages" describe the interface to the service in enough detail for someone to write an application to use the Web service. The way services are defined is through a UDDI document called a Type Model or tModel. In many cases, the tModel contains a WSDL file that describes a SOAP interface to an XML Web service, but the tModel is flexible enough to describe almost any kind of service.

The UDDI directory also includes several ways to search for the services you need to build your applications. For example, you can search for providers of a service in a specified geographic location or for business of a specified type. The UDDI directory will then supply information, contacts, links, and technical data to allow you to evaluate which services meet your requirements.

UDDI allows you to find businesses you might want to obtain Web services from. What if you already know whom you want to do business with but you don't know what services are offered? The [WS-Inspection specification](http://msdn.microsoft.com/en-us/library/ms951237.aspx) allows you to browse through a collection of XML Web services offered on a specific server to find which ones might meet your needs.