

## **WEB SERVICE ENHANCEMENT**

**<http://msdn.microsoft.com/en-us/library/aa529300.aspx>**

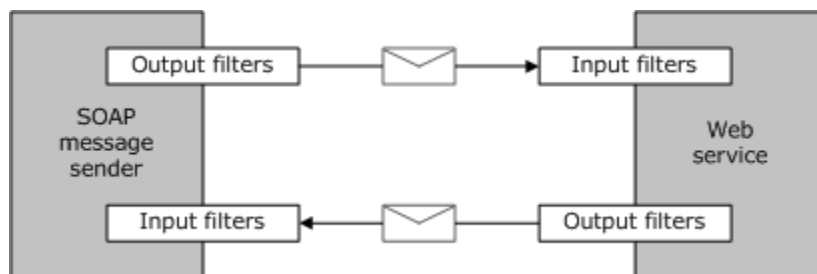
Web Services Enhancements for Microsoft .NET (WSE) is a .NET class library for building Web services using the latest Web services protocols, including WS-Security, WS-SecureConversation, WS-Trust, and WS-Addressing. WSE allows you to add these capabilities at design time using code or at deployment time through the use of a policy file.

The topics in this section describe the concepts and techniques that allow you to build Web services using the latest Web services protocols.

**<http://msdn.microsoft.com/en-us/library/aa529139.aspx>**

### **WSE Architecture**

At its heart, WSE is an engine for applying advanced Web service protocols to SOAP messages. This entails writing headers to outbound SOAP messages and reading headers from inbound SOAP messages. It may also require transforming the SOAP message body — for instance, encrypting an outbound message's body and decrypting an inbound message's body, as defined by the WS-Security specification. This functionality is encapsulated by two sets of filters, one for outbound messages and one for inbound messages. All messages leaving a process — request messages from a client or response messages from server — are processed using the outbound message filters. All messages arriving in a process — request messages to a server or response messages to a client — are processed using the inbound message filters. The following diagram shows this simple architecture.



WSE filter chains are integrated with the SOAP Messaging built-into WSE, as well as the ASP.NET Web services infrastructure.

### **Integration with ASP.NET Web Service Proxies (Sender-side)**

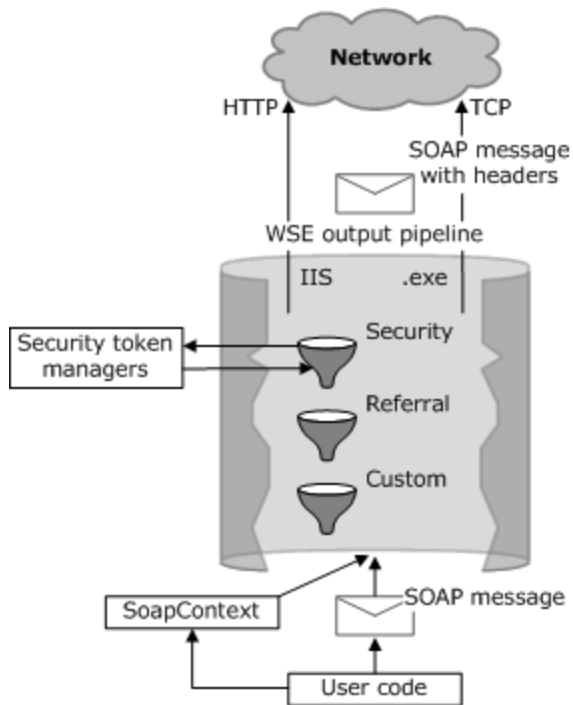
WSE input and output filters are exposed to ASP.NET Web services clients through a proxy base class called **WebServicesClientProtocol**. **WebServicesClientProtocol** extends the default base class for Web service proxies, **System.Web.Services.SoapHttpClientProtocol**. The new proxy base class ensures that WSE filters have a chance to process the SOAP messages that are exchanged whenever a client sends a SOAP message. In order to use WSE, you need to change the base class of each of your proxies to **WebServicesClientProtocol**. The easiest way to do this is to use the WSE Settings tool. For more details, about using WSE Settings tool, see WSE Settings 3.0 Tool.

The **WebServicesClientProtocol** proxy base class is implemented using two communication classes from the standard **System.Net** communication classes: **WebRequest** and **WebResponse**. At a high-level, the **WebRequest** instance sends SOAP requests and the **WebResponse** instance receives SOAP responses.

The **WebRequest** instance parses a request stream containing a SOAP message into an instance of the **SoapEnvelope** class, an extension of **System.Xml.XmlDocument**, the standard .NET DOM API. Then it passes the request through the chain of output filters. Each filter has the chance to modify the request data any way it likes. Often, a filter simply adds protocol headers, but in some cases they modify the body of a message too, such as when the SOAP body is encrypted.

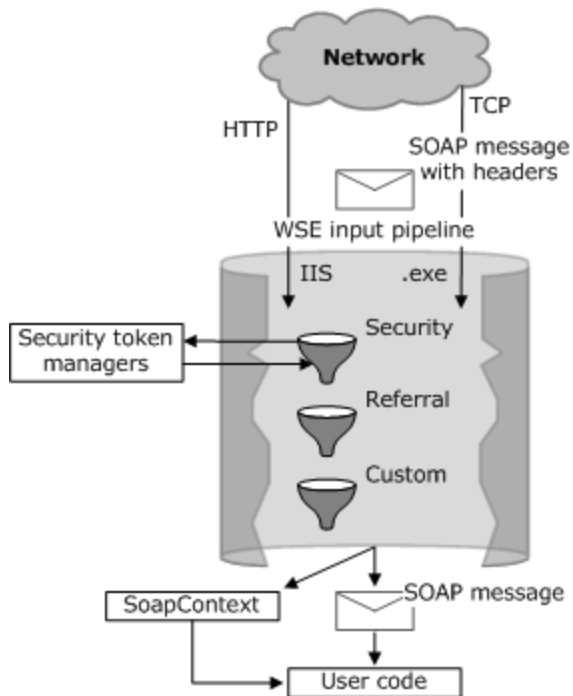
The set of filters and their behavior is controlled through the policy associated with the SOAP message. This policy is comprised of one or more ordered policy assertions, each of which defines a set of input and output filters for a SOAP message exchange between a client and a Web service. Each policy assertion's output filter is passed the **SoapEnvelope** in the order specified by the policy.

The following diagram illustrates how SOAP requests are passed through a set of filters.



The **WebResponse** operates on the SOAP message the opposite of the **WebRequest** instance. It parses a response stream containing a SOAP message into an instance of the **SoapEnvelope** class and passes it through the chain of input filters. Each filter has the chance to examine and modify the response data any way it likes. Input filters check the validity of protocol headers and modify the contents of the message's body as needed to undo the work of an output filter (decryption, for example). Like **WebRequest**, the set of filters are defined by the policy associated with the SOAP request.

The following diagram illustrates how SOAP responses are passed through a set of filters.



The **WebServicesClientProtocol** encapsulates the use of the **WebRequest** and **WebResponse** classes so that you don't have to deal with them directly. However, you still need a way to get the protocol properties for both outbound and inbound messages. To that end, the **WebServicesClientProtocol** class exposes two properties, **RequestSoapContext** and **ResponseSoapContext**, both of type **SoapContext**. These objects reflect the protocol properties of the next message to be sent and the last message that was received, respectively.

### Integration with ASP.NET Web Services (Receiver-side)

WSE input and output filters are exposed to ASP.NET Web services through a server-side SOAP protocol factory, **WseProtocolFactory**. The goal of the new SOAP protocol factory is to ensure that WSE filters have a chance to process the SOAP messages that are exchanged whenever a Web service's methods are invoked.

The **WseProtocolFactory** class copies inbound messages into memory streams. It processes them using WSE input filters before the message is deserialized into input parameters for the target method. The extension also sets up a memory stream for the target method to serialize its output parameters into. After that serialization step occurs, the output message is passed through WSE output filter and then sent on its way.

For the **WseProtocolFactory** to run for a Web service, the **WseProtocolFactory** must be configured in the Web.config file for the Web service. Specifically, add the `<soapServerProtocolFactory>` Element to the Web.config file in the virtual directory where the Web service is deployed.

### Sending and receiving SOAP Messages using TCP

WSE enables you to send and receive SOAP messages using the TCP protocol. This can be accomplished with or without an HTTP server, making it possible to write extremely flexible and lightweight Web services. WSE supports both a one-way messaging model and a request/response pair model. One-way messaging is accomplished using the SoapSender and SoapReceiver classes, whereas both one-way messaging and request/response messaging can be accomplished using the SoapClient and SoapService classes.

**<http://msdn.microsoft.com/en-us/library/aa529259.aspx>**

## **Features of WSE**

The Web Services Enhancements for Microsoft .NET (WSE) has many features to augment applications built on Web services using the .NET Framework 2.0.

### **In This Section**

<b>Feature</b>	<b>Description</b>
Securing Applications That Use Web Services	Explains the security features of WSE, such as digital signatures, encryption, and authentication.
Integration with .NET Framework 2.0 and Visual Studio 2005	Explains how WSE integrates with the .NET Framework 2.0 and Visual Studio 2005.
Sending Large Amounts of Data in a SOAP Message Using WSE	Explains how to send large amounts of data, such as a binary file, in a SOAP message.
Routing SOAP Messages Using WSE	Explains how SOAP messages can be routed through intermediaries prior to reaching their ultimate destination.
Hosting ASP.NET Web Services Outside of IIS	Explains the types of applications that WSE enables ASP.NET Web services to be hosted in outside of IIS.
Sending and Receiving SOAP Messages Using WSE SOAP Messaging	Explains the alternative method provided by WSE to send and receive SOAP messages using the SoapClient and SoapService API.

**<http://msdn.microsoft.com/en-us/library/aa529262.aspx>**

## **Securing a Web Service**

For the majority of cases, securing Web services using the Web Services Enhancements for .NET (WSE) is done by declaratively stating the security requirements for incoming and outgoing SOAP messages in an XML file. These requirements, collectively known as a *policy*, are defined either by (a) using the WSE Settings 3.0 Tool tool and its Policy tab

from within Visual Studio 2005 or (b) manually adding a <policy> Element (Policy) to the XML file. Whether the tool is used or not, each **<policy>** element has a **name** attribute that uniquely names the policy. Once the policy is declared in the XML file, which is known as a *policy file*, the policy is applied to a Web service method by applying a PolicyAttribute attribute with a policy name to the class that is implementing the Web service method.

Alternatively, the policy for a Web service can be specified in code when the deployment environment is known ahead of time and is not likely to change. Typically, it is more flexible to allow an administrator to define the policy for an application when it is deployed using a policy file, but WSE does allow you to specify the policy in code. To specify the policy in code, a PolicyAttribute attribute is still applied to the class that is implementing the Web service method, but a type is provided instead of a policy name. That type must derive from Policy, and in its constructor the security requirements are specified. For more details, see *How to: Secure a Web Service Without Using a Policy File*.

The following sections describe how to create a policy file in more detail when you are not using the WSE Settings 3.0 Tool tool or you are modifying the contents of the policy file that is generated by the tool.

## **In This Section**

<b>Topic</b>	<b>Description</b>
Policy Files	Describes what a policy file is and its basic XML structure.
Policy Assertions	Describes how a policy assertion is used to describe the requirements for a SOAP message exchange.
Policy Extensions	Describes the extensibility points in the policy framework.
Turnkey Security Assertions	Details the turnkey security assertions that ship with WSE.