SOAP nodes in IBM WebSphere Message Broker V6.1, Part 1: SOAP node basics

Rob Henley
Matthew Golby-Kirk

19 June 2008

SOAP nodes send and receive SOAP-based Web services messages, allowing a message flow to interact with Web service endpoints. The messages might be plain SOAP, SOAP with Attachments (SwA), or Message Transmission Optimization Mechanism (MTOM). The nodes are configured using Web Services Description Language (WSDL) and support WS-Security and WS-Addressing. This four-part series describes the SOAP nodes, the logical tree for the new SOAP domain, and details of configuration and runtime behavior. In this first article, you learn about the basic use of the nodes. You should have a general familiarity with SOAP-based Web services and WSDL to follow along with this article series.

View more content in this series

Introduction

SOAP nodes are used together to implement common Web services scenarios and are generally the best choice for new message flows working with Web services. SOAP nodes, which are configured using WSDL 1.1, support SOAP-based Web services, sending and receiving messages that can be plain SOAP (version 1.1 or 1.2), SwA, or MTOM (see the terminology list for definitions of terms used in this article). They combine message transport and SOAP semantics to support a consistent SOAP domain logical tree format and the next-generation Web services standards WS-Security and WS-Addressing.

Transport-specific nodes, such as HTTPInput and HTTPReply, are still useful in certain circumstances. For instance, you can use them if you have existing flows using these nodes and you don't foresee any requirement for WS-Addressing or WS-Security support, or you don't have a WSDL definition and don't intend to create one—maybe because you're using a non-SOAP-based Web service technology like XML Remote Procedure Call (XML-RPC) or Representational State Transfer (REST). But if you're creating a new message flow and using WSDL, then the SOAP nodes offer a more complete solution.

The SOAP nodes

The SOAP nodes use the new SOAP parser and SOAP logical tree format. The new nodes are:

- **SOAPInput** and **SOAPReply:** Used together to provide (that is, implement) a Web service.

- **SOAPRequest:** Used to invoke a Web service.
- **SOAPAsyncRequest** and **SOAPAsyncResponse:** Used together to invoke a Web service asynchronously (which simply means that the message flow is not blocked at the SOAPAsyncRequest while waiting for the response).

There are two basic Web services scenarios, which are covered in the next sections: provider and consumer.

Provider scenario

Your message flow provides or implements a Web service. Of course, you can implement a simple Web service entirely within a message flow. However, a more typical provider scenario is one in which the message flow delegates some of the work to one or more external agents. For example, a message flow can do one of the following:

- Interrogate or update a database.
- Delegate some business processing to an existing application.

In some cases, such a message flow can be said to provide a new Web service interface to an existing application. However, note that the message flow can also enhance, restrict, or otherwise adapt the capabilities of the existing application, or compose the capabilities of multiple applications and other data repositories.

In Figures 1 and 2 you can see two styles of Web service provider flow where an existing application is invoked. These examples show the existing application being invoked over IBM® WebSphere® MQ, but you can use other transports (in fact, the existing application can be another Web service, which is covered in the Consumer scenario section).

Figure 1. Provider with synchronous implementation (over WebSphere MQ)

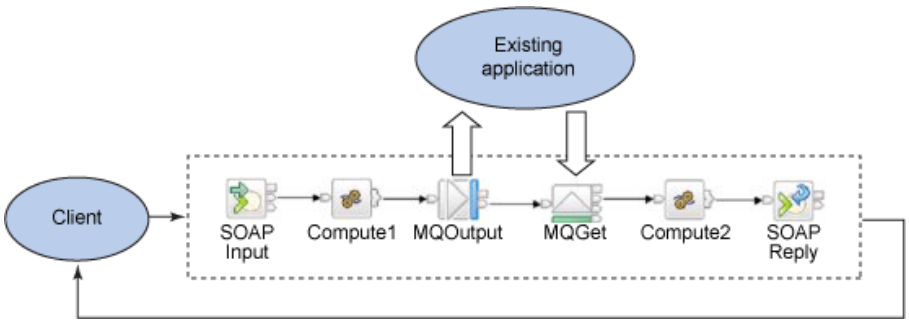
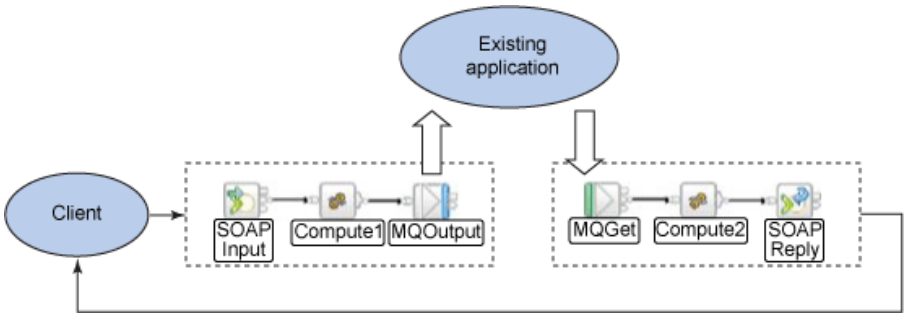


Figure 2. Provider with asynchronous implementation (over WebSphere MQ)



In these examples, the Compute nodes transform the SOAP messages to and from the required external application format. You can also use Mapping nodes or Java™ Compute nodes.

In Figure 1, a single transaction handles the request-reply translation. This supports simple rollback and recovery, but can impact performance if the application being invoked can't respond quickly.

In Figure 2, the SOAPInput-MQOutput flow (the request flow) completes after sending the message to the application. This flow is then available to handle further requests before the application being invoked has responded. In this scenario, a correlation context must be saved in the request flow and restored in the reply flow (see the Correlation in the provider scenario section later in this article).

Some common features of the provider scenario are:

- The SOAP nodes listen on port 7800 (HTTP) or 7843 (HTTPS) by default.
- The SOAPInput node can accept any operation defined by the configured WSDL binding (see the Use WSDL to configure SOAP nodes section).
- The WSDL might be generated from an existing message set modelling the interface to the external application, or an existing WSDL might be available for import.
- If a SOAP processing error occurs at the SOAPInput node, it can return a SOAP fault either directly to the sender or to a different endpoint if directed to do so by WS-Addressing.
- You can wire the Catch and/or Failure terminals and build your own SOAP fault messages to be returned by a SOAPReply node, again, as directed by WS-Addressing if this is used.
- The SOAPInput node can return a SOAP fault to the client to indicate that a timeout has occurred. This happens if the SOAPReply node hasn't sent the response message within the period specified by the property *Maximum client wait time (sec)** on the HTTP Transport tab of the SOAPInput node Properties.

Part 4 of this series will describe scalability, error handling, and WS-Addressing in more detail.

Consumer scenario

Your message flow invokes a Web service. A common consumer scenario is a message flow that integrates some existing business logic available as a Web service. A specific variation of this scenario is that the message flow also provides a Web service. This is sometimes known as a *facade*, because one Web service is effectively re-exposed with a slightly different interface or with some additional processing being done.

Figures 3 and 4 illustrate a facade scenario.

Figure 3. Consumer: synchronous (facade)

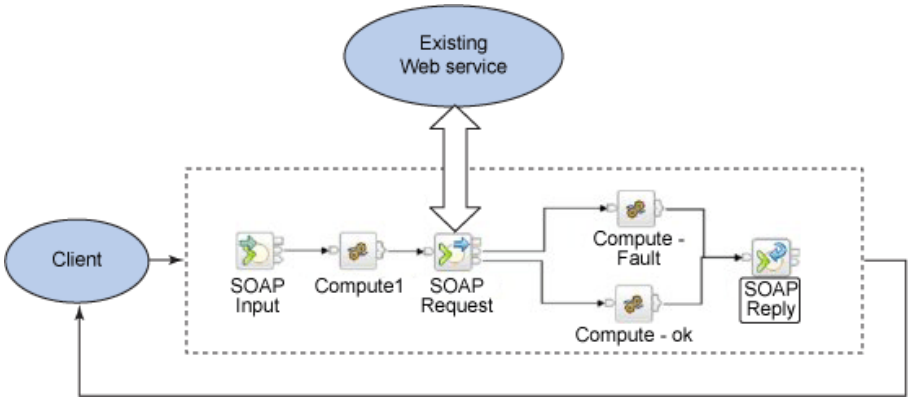
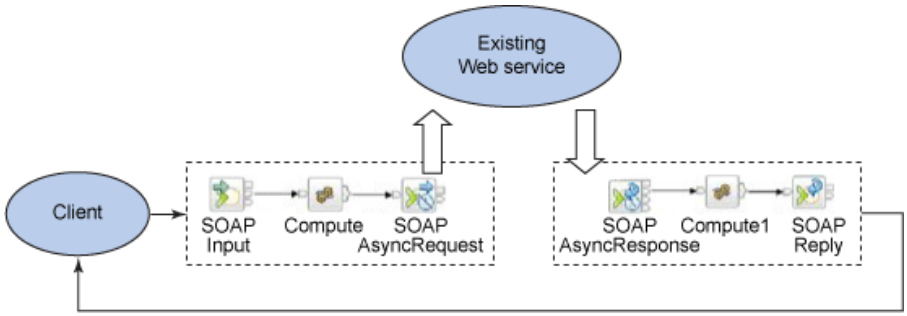


Figure 4. Consumer: asynchronous (facade)



Terminology

- **W3C:** The World Wide Web Consortium, the body that publishes the Web services standards.
- **SOAP:** The W3C standard XML message format for Web services messages.
- **SOAP with Attachments (SwA):** The W3C standard for Web services that need to incorporate attachments, such as image data, in their messages.
- **SOAP domain:** The broker domain for working with Web services messages. The messages are represented in a message flow using the **SOAP domain logical tree**.
- **Web Services Description Language (WSDL):** The W3C standard for describing a Web service.
- **Deployable WSDL:** The broker WSDL representation used to configure SOAP domain nodes. All the schema definitions for deployable WSDL are held in broker message definitions. Deployable WSDL can be created by importing regular WSDL definitions. Likewise, deployable WSDL can be exported as regular WSDL for external consumption.
- **Multipurpose Internet Mail Extensions (MIME):** A message format for multipart messages and the underlying format for SwA and MTOM.
- **Message Transmission Optimization Mechanism (MTOM):** The use of MIME to optimize the bitstream transmission of SOAP messages that contain significantly large base64Binary elements.
- **WS-Security:** An Organization for the Advancement of Structured Information Standards (OASIS) specification that describes how to apply security standards to SOAP, letting you, for instance, authenticate a client and encrypt or sign all or part of a SOAP message.
- **WS-Addressing:** A W3C specification that describes how to specify identification and addressing information for messages. It provides a correlation mechanism and lets more then two services interact.

- **mustUnderstand header:** A SOAP header marked with a `mustUnderstand` attribute set to `1` (as specified for SOAP 1.1 and the WS-I Basic Profile) or `true` (SOAP 1.2).
- **XML Remote Procedure Call (XML-RPC):** An alternative XML-based message format for non-SOAP Web services. (See Resources for a link to more information.)
- **Representational State Transfer (REST):** An HTTP-based alternative to XML and SOAP for Web services.

In these examples, the Compute nodes make any adjustments required to the SOAP messages. You can also use Mapping nodes or Java Compute nodes. In Figure 3, the message flow makes a synchronous request. The flow is blocked until the response is received or times out, which can have performance implications.

In Figure 4, the request is asynchronous. The first half of the flow completes after the request is sent, allowing it to dispatch further requests without undue delay. (Timely dispatch can also be achieved in the synchronous case, as shown in Figure 3, by configuring additional instances of the flow, but this consumes more broker resources and is therefore less scalable.)

The response is received by the SOAPAsyncResponse node, typically in a different message flow. This node must be in the same execution group as the SOAPAsyncRequest node, but it uses a separate thread.

Some common features of the consumer scenario are:

- The WSDL for the external service typically exists and is available for import.
- The SOAPRequest (or SOAPAsyncRequest) configuration must specify a particular WSDL operation and a binding (see the Use WSDL to configure SOAP nodessection).
- The SOAPRequest (or SOAPAsyncResponse) node can receive either a valid Web service response or a SOAP fault. Typically, you connect the fault terminal to handle faults as a special case, as shown in Figure 3.

The SOAPAsyncRequest and SOAPAsyncResponse nodes always use WS-Addressing. Part 4 of this series will describe WS-Addressing in more detail, but there's one important point worth mentioning now: If you create a message flow, as shown in Figure 4, then the existing Web service must support WS-Addressing. In particular, if you implement the existing Web service using a SOAPInput - SOAPReply flow, then you must select **Use WS-Addressing** on the WS Extensions tab of the SOAPInput node.

Use WSDL to configure SOAP nodes

You use WSDL to configure SOAP nodes to define:

- The messages to be sent and received.
- The endpoint details.

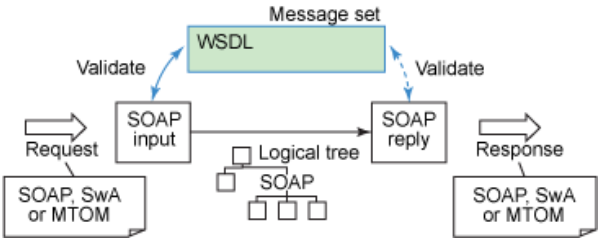
You can then further configure the nodes to override endpoint details (if necessary) to configure any use of WS-Addressing, WS-Security, and SOAP mustUnderstand headers, and to modify other properties relating to parsing (such as validation) or transport (for instance, you might need to configure the use of the HTTPS transport to use SSL in conjunction with WS-Security and UsernameToken).

Figures 5 and 6 show how to use this WSDL to configure the SOAP nodes introduced in the previous section. Part 2 of this series will describe the SOAP logical tree. Part 3 will describe where the WSDL comes from and give more detail about individual configuration settings.

For now, just assume that you have a WSDL definition available as `Deployable WSDL` in a broker message set. **Note:** Only SOAPInput, SOAPRequest, and SOAPAsyncRequest are configured explicitly using WSDL. SOAPReply and SOAPAsyncResponse inherit the definition supplied on their partner nodes.

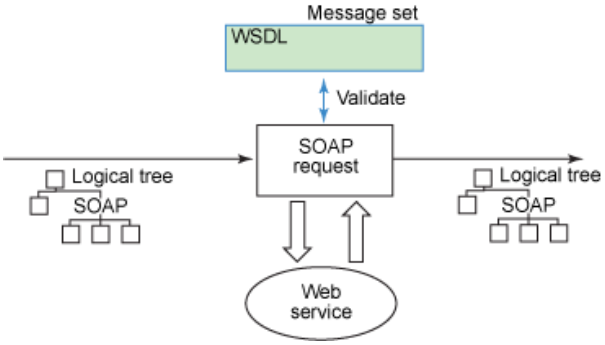
In Figure 5 you can see that the WSDL definition in the message set is used to configure the SOAPInput node. The same definition is automatically applied to the SOAPReply node. The Web service request and response (the input and output messages) are validated against the WSDL at run time. A SOAP logical tree represents the Web services message in the flow.

Figure 5. SOAPInput - SOAPReply configuration



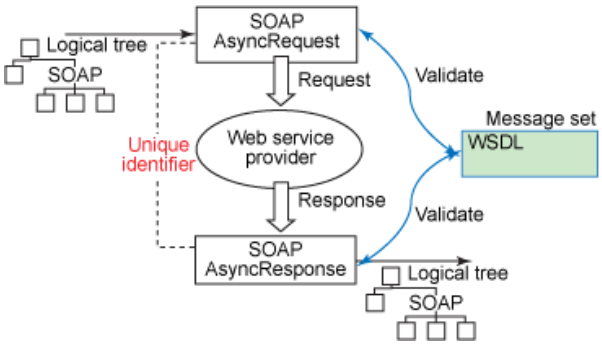
In Figure 6 the WSDL definition in the message set is used to configure the SOAPRequest node. The Web service request and response messages are both validated against the WSDL at run time.

Figure 6. SOAPRequest configuration



In Figure 7 the WSDL definition in the message set is used to configure the SOAPAsyncRequest node. The same definition is automatically applied to the SOAPAsyncResponse node. The Web service request and response are both validated against the WSDL at run time. You associate the two nodes by configuring them both with the same unique identifier: a URL fragment specified on the Basic tab of the node Properties.

Figure 7. SOAPAsyncRequest - SOAPAsyncResponse configuration



Correlation in the provider scenario

The SOAPInput node and the SOAPReply node are designed to be used together. The SOAPReply node must always be in the same execution group as its corresponding SOAPInput node.

Often the two nodes are in the same message flow, in which case the SOAPReply node automatically detects which request it's replying to. Otherwise, a `ReplyIdentifier` in the LocalEnvironment tree is used to let you correlate the inbound SOAP message with the corresponding reply sent from the SOAPReply node.

As a special case, even if the two nodes are in separate message flows, you might not need to do anything specific if these flows use the SOAPAsyncRequest and SOAPAsyncResponse nodes, as shown in Figure 4. In this case the `ReplyIdentifier` is automatically flowed from the request flow to the response flow in the WS-Addressing headers, as long as the `ReplyIdentifier` is in the LocalEnvironment when the AsyncRequest node sends the message.

However, in general when the two nodes are in separate message flows (for example, Figure 2), you need to forward the necessary correlation information to the reply flow yourself. The SOAPInput node sets the `ReplyIdentifier` in the LocalEnvironment at this location: `LocalEnvironment.Destination.SOAP.Reply.ReplyIdentifier`. You need to save this value somewhere and then set it in the reply flow at some point before the SOAPReply node.

One way to save the `ReplyIdentifier` between flows is to store the value on a queue, then use an MQGet node in the reply flow to retrieve it. Other possibilities include saving it in a database or an WebSphere MQ RFH2 header.

Conclusion

This article, Part 1 of the series, described what the SOAP nodes are, how to configure them using WSDL, and how to use them in the basic Web service scenarios. Part 2 covers the SOAP logical tree and its use.