SOAP nodes in IBM WebSphere Message Broker V6.1, Part 2: The SOAP domain logical tree

Rob Henley
Matthew Golby-Kirk

10 July 2008

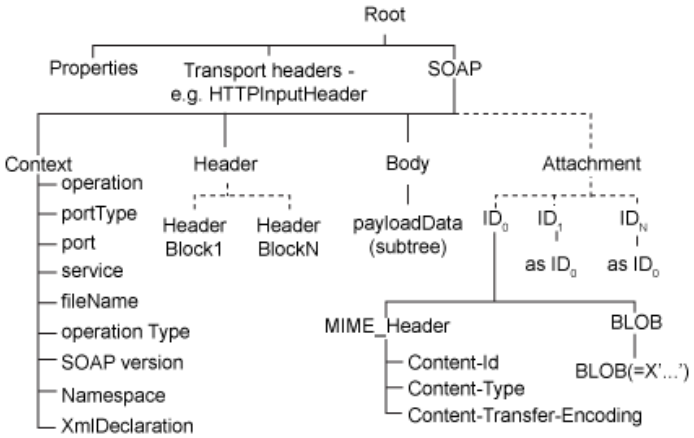
The first article in this four-part series covered the basic use of SOAP nodes, which send and receive SOAP-based Web services messages, allowing a message flow to interact with Web service endpoints. This article, Part 2, describes the new logical tree format used by the SOAP domain. You should have a general familiarity with SOAP-based Web services and WSDL to follow along with this article series. **Note:** This article relates to IBM® WebSphere® Message Broker V6.1 Fix Pack 6.1.0.2. Some details could differ slightly from the 6.1 GA version.

View more content in this series

Introduction to the SOAP domain logical tree

The SOAP nodes described in Part 1 of this series provide a consistent logical tree representation for all Web services messages. At run time the SOAP domain logical tree shown in Figure 1 is used to represent all Web services message formats.

Figure 1. SOAP domain tree



Supported message formats

- SOAP 1.1
- SOAP 1.2
- SOAP with Attachments (SwA)

- Message Transmission Optimization Mechanism (MTOM)

The tree format gives the user a convenient way to access key parts of the message: the SOAP body, SOAP headers, and any SwA attachments (see the terminology list for definitions of terms used in this article series). The message is presented in a consistent way that's independent of the bitstream message format.

The `SOAP` element in the tree corresponds loosely to the SOAP envelope in a Web services message. The `Header` and `Body` elements represent the header and body (payload) sections of the SOAP message. Contextual information, such as namespace attributes and WSDL-derived information, are saved beneath `Context`. Any SwA attachments appear under `Attachment`. (MTOM "attachments" are represented inline as part of the SOAP content, as described in the MTOM section.)

Working with the SOAP tree

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.

A SOAP domain tree (see Figure 1) is automatically *created* in one of these two scenarios:

- A SOAPInput node receives a Web service request.
- A SOAPRequest or SOAPAsyncResponse node receives a Web service response.

A SOAP domain tree must be *supplied* when a message flow uses a SOAPReply, SOAPRequest, or SOAPAsyncRequest node to send a Web service message (although there are some exceptions to this rule; see the Message domain section).

In the case of a SOAPInput-SOAPReply flow, the SOAP domain tree created by the SOAPInput node is typically modified to create the required Web service response message.

In the case of SOAPRequest or SOAPAsyncRequest, the message flow may need to create a SOAP domain tree from scratch. This is done using any of the broker Transformation nodes. The following sections show examples using extended SQL (ESQL) for a Compute node.

Message domain

SOAP nodes are unusual in that they are associated with a specific domain: the SOAP domain. If you look at the Message domain property on the SOAPInput and SOAPRequest nodes, you can see that it's preselected as `SOAP`. This is because the SOAP nodes effectively *embody* the semantics associated with SOAP, such as WS-Addressing and WS-Security support.

Consequently, the SOAP nodes typically work with the SOAP domain tree, as described in the rest of this article. However, there are two exceptions:

- A *non-SOAP domain* tree can be supplied to a SOAPReply, SOAPRequest, or SOAPAsyncRequest node. Particularly in the SOAPRequest or SOAPAsyncRequest case, you might have a non-SOAP message tree representing the SOAP request data. As a convenience, you can pass such a tree directly to the node. Perhaps the most likely example is an XMLNSC tree. In this case, if the root element of the XMLNSC tree is a correctly formed SOAP envelope, then the message is used as is. Otherwise the root element is treated as the SOAP body payload and an appropriate envelope is added automatically. This can be very convenient, but note that the SOAP domain tree in general gives you more flexibility and control.
- A SOAP domain tree can be supplied to a *non-SOAP output node*. If a SOAP domain tree is sent to a non-SOAP output node, an XML serialization of the tree is currently written, rather than a SOAP-style Web service message. This mechanism allows the SOAP domain to be used in conjunction with the Aggregation and Collector nodes, which need the ability to serialize and recreate a logical tree. Ordinarily you don't see this XML-serialized form unless you explicitly route a SOAP domain message to a non-SOAP output node, such as MQOutput. Such a message is probably of limited use, although it can be read back in at a corresponding input node, in this case an MQInput node with the SOAP domain selected.

ESQL namespace declarations and code completion

Usually your ESQL uses at least two namespaces: the namespace for the SOAP envelope and one or more namespaces used by your application data. Appropriate namespace declarations are added to the ESQL automatically if you use the content-assist feature of the ESQL editor. (An ESQL namespace declaration corresponds closely to the idea of a namespace prefix in XML. See the SOAP.Context section for information on how to control the namespace prefixes used when the tree is written out.) In the following examples, the application data uses a `http://a/b/c/myNamespace` namespace, and you assume the following namespace declaration has been made:

```
DECLARE ns1 NAMESPACE 'http://a/b/c/myNamespace'
```

Also, assume a similar declaration, `soapenv`, has been made for the appropriate SOAP envelope namespace. If the WSDL specifies a SOAP 1.1 binding, then this would be:

```
DECLARE soapenv NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/'
```

Note that development time content-assist and mapping node support is enabled automatically when a WSDL definition is imported or generated in a message set. The message definitions created define both the SOAP domain tree itself and the application-specific data that can appear within it. Content-assist is also known as *code completion*. In practice it means that you can type something like `SET OutputRoot.SOAP`, then press `Ctrl+Space` and have the ESQL editor offer you a choice of valid elements that can appear at this position in the path. (If code completion doesn't appear to be working, check that the Default message domain for your message set is SOAP and that your message set project is selected under **Properties > Project References** on your message flow project.)

SOAP.Body

A SOAP message has an XML root tag, `soapenv:Envelope`. The content or payload of a SOAP message appears below Envelope as `soapenv:Body` and is represented as SOAP.Body in the logical tree. If you're building a SOAP tree in your message flow, SOAP.Body is the only part of the tree that *must* be provided. Note that SOAP.Body is a logical path in the SOAP domain tree. It's the same regardless of which version of SOAP was used. In particular, these elements in the logical tree are not namespace qualified.

The following example shows a SOAP response being constructed, in this case by modifying a value received in the corresponding SOAP request (that is, assume the message flow is SOAPInput - Compute - SOAPReply):

```
SET OutputRoot.SOAP.Body.ns1:responsePayload.X=InputRoot.SOAP.Body.ns1:requestPayload.Y
```

Note that:

- The SOAPInput node was configured using a WSDL binding that defines one or more WSDL operations.
- The XML element `ns1:requestPayload` identifies one of these operations, let's call it `op1`.
- The WSDL specifies that the response for `op1` must be `ns1:responsePayload`.
- The WSDL also includes XML Schema definitions that define the detailed structure of `ns1:requestPayload` and `ns1:responsePayload`.

SOAP.Header

A SOAP envelope also contains an (optional) `soapenv:Header`. Any children of `soapenv:Header` are known as header blocks and are held below SOAP.Header in the logical tree. Header blocks may or may not be defined by the XML Schema in your WSDL.

The following example shows how to propagate all the header blocks from an existing SOAP tree:

```
SET OutputRoot.SOAP.Header = InputRoot.SOAP.Header
```

Alternatively, you may want to propagate selected headers; in the following example the header block is `ns1:header1`:

```
SET OutputRoot.SOAP.Header.ns1:header1 = InputRoot.SOAP.Header.ns1:header1
```

In both cases you may also want to add some new headers; in the following example the header block is `ns1:newHeader`:

```
SET OutputRoot.SOAP.Header.ns1:newHeader = '42'
```

Finally, the following example shows how to add an attribute to this header (in the initial release of WebSphere Message Broker V6.1 you need to use the `XMLNSC` keyword instead of `SOAP` to qualify the element type):

```
SET OutputRoot.SOAP.Header.ns1:newHeader.(SOAP.Attribute)soapenv:mustUnderstand = '1'
```

Note: `mustUnderstand` headers will be described in Part 4 of this series.

SOAP.Attachment (SwA)

As mentioned earlier, the `SOAP` element in the tree corresponds loosely to the SOAP envelope. A key difference is that the SOAP logical tree *also* contains any attachments for your Web service message. This means that the logical tree structure is the same, regardless of whether the bitstream for the message is SwA, MTOM, or plain SOAP.

Each SwA attachment is saved under its `Content-Id`. For instance, a Multipurpose Internet Mail Extensions (MIME) part with a header `Content-Id=`**`Id1`** is saved under `SOAP.Attachment.`**`Id1`**. The complete set of MIME headers for the attachment is stored under the subfolder `MIME_Headers`, and the data itself is represented as a `BLOB`.

The following example shows how to propagate all the attachments from an existing SOAP tree:

```
SET OutputRoot.SOAP.Attachment = InputRoot.SOAP.Attachment
```

Alternatively you may want to propagate selected attachments, in this example `Id1`:

```
SET OutputRoot.SOAP.Attachment.Id1 = InputRoot.SOAP.Attachment.Id1
```

In both cases you may also want to add some new attachments. Let's assume you want to add an attachment under a new Content-Id, `Id2`. The following example is somewhat artificial but shows one way of setting the attachment data; in this case the binary data happens to correspond to the ASCII characters <HelloWorld/>:

```
SET OutputRoot.SOAP.Attachment.Id2.BLOB = X'3c48656c6c6f576f726c645c3e'
```

The resulting bitstream is a MIME message. The top-level Content-Type is generated automatically with a value like this:

```
multipart/related; boundary=WMBMIME...; type="text/xml";start="<...>";charset=utf-8
```

The MIME part identified by the `start` parameter (usually the first MIME part) contains the SOAP message. A second MIME part is generated for the new attachment `Id2`. The Content-Type for this part defaults to `application/octet-stream`, but you can specify a value explicitly as follows:

```
SET OutputRoot.SOAP.Attachment.Id2.MIME_Headers."Content-Type" = 'text/xml'
```

Another header that you might want to set explicitly is Content-Id:

```
SET OutputRoot.SOAP.Attachment.Id2.MIME_Headers."Content-Id" = 'Id3'
```

As described earlier, the default Content-Id value is the name of the child of Attachment. In this case, the default Content-Id is `Id2`, but this is overridden by the ESQL, and the value `Id3` is used instead.

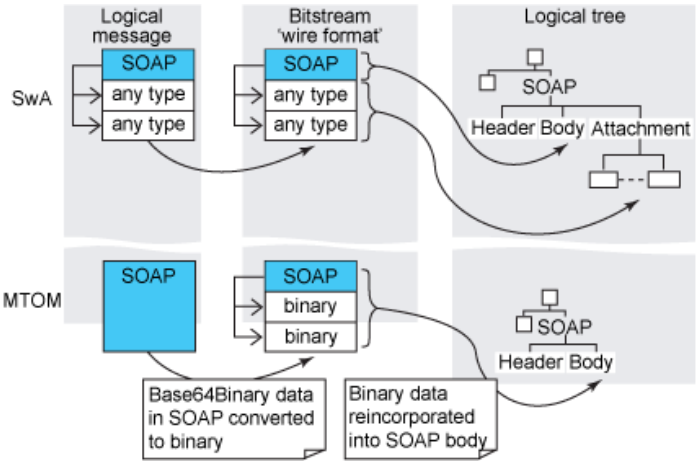
On output, if a SOAP domain tree has any children below SOAP.Attachment, then the bitstream is written as MIME (specifically as SwA), with the attachments appearing as separate MIME parts. In this case it doesn't matter whether the WSDL binding specified the use of MIME or not.

MTOM

MTOM appears to be similar to SwA, in that both use MIME. However, MTOM only uses MIME as an optimisation mechanism, allowing binary data to be sent unencoded (and therefore unexpanded) *on the wire*.

Data that would otherwise have to be encoded in the SOAP message is instead transmitted as raw binary data in a separate MIME part. A large chunk of binary data takes up less space than its encoded representation, so MTOM can reduce transmission time (although it incurs some processing overhead). Candidate elements to be transmitted using MTOM are defined as base64Binary in the WSDL (XML Schema). In SwA, attachments are treated as logically separate items. In MTOM, the elements always remain integral parts of the SOAP message; they just happen to be split out temporarily while the message is "in flight."

Figure 2. SwA and MTOM comparison



The SOAP domain handles inbound MTOM messages automatically. You won't see any evidence of MTOM in the SOAP domain tree though; the MTOM parts will already have been reincorporated into the SOAP Body.

The use of outbound MTOM messages can be configured on the SOAPRequest, SOAPAsyncRequest, and SOAPReply nodes. An MTOM output message is written if all of the following are true:

- The **Allow MTOM** option is checked on the WS Extensions tab of the node.
- **Validation** is enabled on the Validation tab of the node.
- There are elements in the output message that are identified as base64Binary in the associated XML Schema and whose length does not fall below a default threshold size of 1000 bytes (let's call these qualifying elements).

In this case the qualifying elements are output as individual MIME parts in an MTOM message. Otherwise MTOM is not used. A user trace message confirms whether MTOM was used. The `Allow MTOM` node option and the default element-size threshold can be overridden in the `LocalEnvironment`.

The ESQL in the following code snippet shows `Allow MTOM` and the element-size threshold being set prior to a SOAPRequest or SOAPAsyncRequest node:

```
SET OutputLocalEnvironment.Destination.SOAP.Request.AllowMTOM = true;
SET OutputLocalEnvironment.Destination.SOAP.Request.MTOMThreshold = 100; -- bytes
```

The equivalent settings required for a SOAPReply node are shown in this example:

```
SET OutputLocalEnvironment.Destination.SOAP.Reply.AllowMTOM = true;
SET OutputLocalEnvironment.Destination.SOAP.Reply.MTOMThreshold = 100; -- bytes
```

Note that SwA and MTOM aren't used together, and the presence of any children below `SOAP.Attachment` always results in the use of SwA exclusively, with no MTOM processing.

The top-level Content-Type for an MTOM message has a value of the form shown here:

```
>multipart/related;boundary=WMBMIME...; type="application/xop+xml";
start="<...>";charset=utf-8
```

SOAP.Context

As described already, WSDL plays the central role in configuring a message flow that uses the SOAP nodes. In fact, some key information derived from WSDL is added to the SOAP.Context part of the tree when the tree is created at run time. For instance, in the example shown under the SOAP.Body section, `operation=op1` would be included under the Context.

This section, however, describes those aspects of the SOAP.Context that affect the *output* bitstream and may, therefore, need to be set explicitly in your message flow.

First, the following example shows how to propagate the whole SOAP Context from an existing tree:

```
SET OutputRoot.SOAP.Context = InputRoot.SOAP.Context
```

The following sections describe the role of individual parts of the SOAP.Context.

Namespaces

One of the key parts of the `SOAP.Context` is the `Namespace` subtree. Often you'll want to specify particular namespace prefixes for the namespace URIs in your messages. (Otherwise prefix names are generated automatically.) When a Web service message is received, appropriate

entries are created automatically under `SOAP.Context.Namespace`, and often no further action is needed. However, if you're creating a SOAP tree from scratch or you need to add or modify namespace prefix definitions, you can do this by adding or modifying the name-value children below `SOAP.Context.Namespace`.

You can use the ESQL in the following example to create the namespace prefix definition for `ns1`:

```
SET OutputRoot.SOAP.Context.Namespace.(SOAP.NamespaceDecl)xmlns:ns1=
'http://a/b/c/myNamespace';
```

Note: Again, in the initial release of WebSphere Message Broker V6.1 you need to use the `XMLNSC` keyword instead of `SOAP` to qualify the element type.

If you're trying to set up a specific namespace prefix and it's not appearing in the bitstream, check the element types. The following code snippet is a part of the output from a Trace node, including the pattern `${Root}` for this example. The element types are the hex strings, which start each line:

```
(0x01000000):Namespace = (
(0x03000102)http://www.w3.org/2000/xmlns/:SOAP-ENV = 'http://.../soap/envelope/'
(0x03000102)http://www.w3.org/2000/xmlns/:ns1 = 'http://a/b/c/myNamespace'
)
```

These correspond to the namespace prefix declarations in a bitstream, shown here:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
xmlns:ns1="http://a/b/c/myNamespace">...
```

SOAP version

If `Context.SOAP_Version` is specified and its value is valid (1.1 or 1.2), then an envelope with the specified version is output. Otherwise the SOAP version specified in the WSDL is used. However, note that if the payload is a SOAP Fault, then it's the user's responsibility to ensure that the Fault message is compatible with the selected SOAP version.

You can use the following ESQL to explicitly set the SOAP version prior to a SOAPRequest or SOAPAsyncRequest node:

```
SET OutputRoot.SOAP.Context.SOAP_Version = '1.2'
```

In this case, note that if validation is enabled for the response, you must ensure that the appropriate SOAP envelope message definition is available in the message set. You can do this by selecting **New > Message Definition File From .../ IBM Supplied Message** from the context menu on the message set, and then selecting the appropriate SOAP envelope message. (When you import or generate your deployable WSDL in the message set, only the SOAP envelope specified by the WSDL binding is imported.)

Mapping the SOAP tree

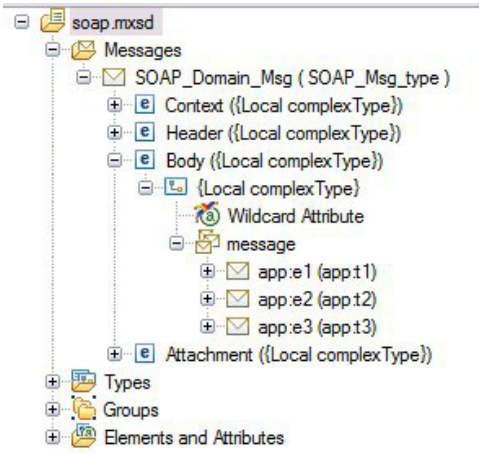
If you use a Mapping node instead of a Compute node to work with the SOAP tree, you typically need to define submaps for any transformation of the SOAP.Header or SOAP.Body. This is because these items in the SOAP tree are defined as wildcards.

A discussion of submaps is outside the scope of this article, but an alternative approach is worth mentioning. You can modify `soap.mxsd` (in the default namespace of your message set) so that it explicitly lists all the elements allowed to appear beneath Header or Body in the message. The changes you need to make to `soap.mxsd` are:

1. Import the mxsd(s) that define the elements you need. If you started off by importing WSDL, the mxsd is a file like `my_InlineSchema.mxsd` under the appropriate namespace. On the Properties tab for soap.mxsd, right-click **Imports**, select **Add Import**, then browse for your mxsd.
2. Add references to the required elements. To add elements that can appear below Body, expand **SOAP_Domain_Msg.Body** in the Outline view (see Figure 3). Right-click **message**, select **Add Message** from the context menu, and select an element to add.

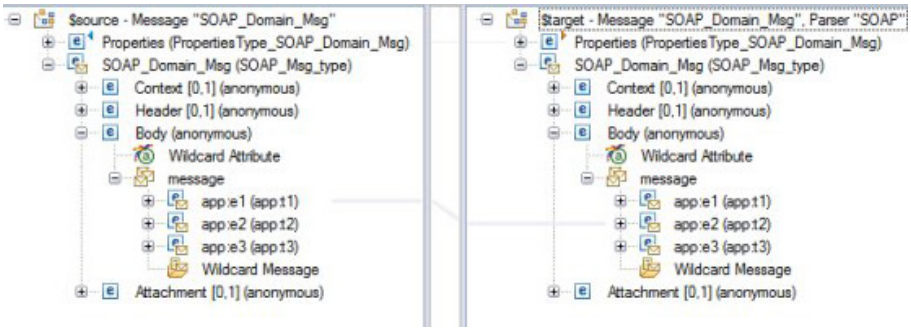
Figure 3 shows soap.mxsd after the addition of elements `e1`, `e2`, and `e3`. In this case, `app` is the namespace prefix allocated to your imported namespace.

Figure 3. Tailored soap.mxsd



You can then use elements `e1` and `e2` as the source or target when defining a mapping, as shown in Figure 4.

Figure 4. Message mapping editor



Conclusion

You've examined the SOAP domain logical tree and how it's used. In Part 3, you'll look at the detailed configuration of the SOAP nodes.