OTA

The OpenTravel™ Alliance

# Transport Protocol Reference

# SOAP

Version 0.2
6 June 2006

## TABLE OF CONTENTS



 

# 1. Introduction

SOAP is a loosely coupled protocol for interaction between applications that may have been developed on different platforms using different languages. SOAP uses XML for payload and a wide range of underlying protocols for transport - one of them being HTTP.

Earlier proprietary protocols required client and server applications to be of the same breed, use a common protocol framework or at least run on the same platform. With the introduction of SOAP, this tight coupling between client and server applications was eliminated and applications could communicate across platform and language barriers.

Many applications are already using SOAP for interchange of OTA documents, and more are in the process of being developed. Up until now there have not been any guidelines or specifications for how OTA documents should be encapsulated and transported in SOAP messages. This has prevented many existing applications from interoperating, even when these applications conform to the SOAP and OTA specifications. This document aims to bridge that gap and acts as guide to software architects and developers that want to create interoperable OTA clients and services using SOAP.

In conjunction with this document, OTA is also releasing a document on WSDL publication. These documents can be used as a set of common guidelines for development and publication of OTA services that are based on SOAP.

## 1.1. Purpose

The main purpose of this document is to provide well defined guidelines for creating and using OTA services over SOAP in a manner that ensures interoperability. The goal is that any party that develops OTA-based services in accordance with this document should be able to interoperate without having to implement any changes in SOAP/OTA message structure or at a protocol level. This document also attempts to outline the simplest method of transmitting OTA messages over SOAP. With simplicity comes ease of implementation and in most cases also efficiency. OTA recommendation of these guidelines will help to ensure wide distribution amongst implementers within the travel industry.

## 1.2. The SOAP Transport Protocol

Microsoft started working on the SOAP specification as early as 1998, and a specification was first released in late 1999. This specification used a sub-set of the then in-progress XML Schema specification. SOAP was initially intended as a replacement for existing RPC protocols such as GIOP/IIOP, DCE/DCOM, RMI, and ONC, but has also developed into a document exchange transport mechanism with similarities to EDI. SOAP was submitted to W3C's XML Protocol Working Group in 2000. SOAP was released as a "Note" from W3C in May 2000.

SOAP provides a simple and extensible vehicle for interchanging data and invoking remote services using XML, as demonstrated by these skeleton SOAP structures:

 

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
   <!-- Routing, security or other control data. -->
 </soap:Header>
 <soap:Body>
   <!-- RPC method call or document data. -->
 </soap:Body>
</soap:Envelope>
```

**Figure 1: Skeleton SOAP Request**

The only limitation to the content of the SOAP Body element is that it must be a valid XML document. This implies that the content is a single XML element, which either represents a remote service/method or is the root element of an XML document that is being exchanged.

The Header element is optional in all SOAP messages and is used to carry information apart from the actual envelope payload – such as routing or security information.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
   <!-- Routing, security or other control data. -->
 </soap:Header>
 <soap:Body>
   <!-- RPC method response or document data. -->
 </soap:Body>
</soap:Envelope>
```

**Figure 2: Skeleton SOAP Positive Response**

The same limitation with regard to the content of the SOAP Body element applies to positive SOAP response messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
   <!-- Routing, security or other control data. -->
 </soap:Header>
 <soap:Fault>
   <faultcode>soap:Client.AppError</faultcode>
   <faultstring>Application Error</faultstring>
   <detail>
     <message>Something went wrong</message>
     <errorcode>12345</errorcode>
   </detail>
 </soap:Fault>
</soap:Envelope>
```

**Figure 3: Skeleton SOAP negative response**

Negative response messages contain a Fault element in place of a Body element.

 

### 1.2.1. The Need for Interoperability

SOAP is already being widely used for transporting OTA documents. However, due to the dual purpose of SOAP (RPC vs. messaging) and SOAP's flexibility with regards to structure, a whole range of SOAP structures are being used to transport OTA data.

Many travel companies expose OTA or OTA-like services using SOAP, with different implementation approaches:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <OTA_CancelRQ xmlns="http://www.opentravel.org/OTA/2003/05" Version="2.001">
     <POS>
       <Source ISOCountry="US" ISOCurrency="NOK" PseudoCityCode="HUR">
         <RequestorID ID="abc.123" URL="abcde..."/>
       </Source>
     </POS>
     <UniqueID ID="DGNJ6012990-389" Type="14"/>
   </OTA_CancelRQ>
 </soap:Body>
</soap:Envelope>
```

**Figure 4: SOAP Messaging**

Using SOAP messaging, the OTA payload is the only and immediate child of the SOAP Body element. This is the simplest and most efficient means of transporting OTA messages over SOAP.

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <acme:otaServiceCancel xmlns:acme="http://www.acme-travel.com">
     <OTA_CancelRQ xmlns="http://www.opentravel.org/OTA/2003/05" Version="2.001">
       <POS>
         <Source ISOCountry="US" ISOCurrency="NOK" PseudoCityCode="HUR">
           <RequestorID ID="abc.123" URL="abcde..."/>
         </Source>
       </POS>
       <UniqueID ID="DGNJ6012990-389" Type="14"/>
     </OTA_CancelRQ>
   </acme:otaServiceCancel>
 </soap:Body>
</soap:Envelope>
```

**Figure 5: SOAP RPC**

With SOAP RPC the immediate child of the SOAP Body element is an XML element that describes the method or function that is being invoked, in this case "otaServiceCancel". The namespace of this element is most often defined by the service application (it is not the OTA namespace).

 

```
Accept-Language: en
Content-Type: multipart/related; type="text/xml"; boundary="----…"
Content-Id: soap-envelope
Content-Length: 923
SOAPAction: OTA_CancelRQ
User-Agent: Acme SOAP Client

-----=Multipart_Boundary_1136385692441_3=----
Content-Type: text/xml
Content-Id: soap-envelope

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <acme:otaServiceCancel xmlns:acme="http://www.acme-travel.com">
     <acme:payload location="cid:OTA_CancelRQ"/>
   </acme:otaServiceCancel>
 </soap:Body>
</soap:Envelope>
-----=Multipart_Boundary_1136385692441_3=----
Content-Type: text/xml; charset="UTF-8"
Content-Id: OTA_CancelRQ

<OTA_CancelRQ xmlns="http://www.opentravel.org/OTA/2003/05" Version="2.001">
 <POS>
   <Source ISOCountry="US" ISOCurrency="NOK" PseudoCityCode="HUR">
     <RequestorID ID="abc.123" URL="abcde..."/>
   </Source>
 </POS>
 <UniqueID ID="DGNJ6012990-389" Type="14"/>
</OTA_CancelRQ>
```

**Figure 6: SOAP RPC with Attachment**

All the above are taken from real-world use of SOAP for interchange of OTA messages. There is nothing inherently wrong with any of these methods, they all use SOAP to carry OTA messages, but they are not interchangeable.

With increased use of OTA messages, an increasing number of SOAP-based OTA services, and with the variety of SOAP structures that can be used to carry OTA messages, the need for a well-defined specification in this area is obvious.

This document outlines a set of guidelines for a uniform method of using SOAP for interchange of OTA messages This specification does not in any way state that OTA services or clients must use SOAP for transport, nor must they conform to this specification. This document is a *guideline* that can be used to maximize interoperability with other SOAP-based OTA clients and services.

## 1.3. Document Scope

This document contains guidelines for interchange of OTA documents/messages using the SOAP protocol.

This document is not intended to contain contradiction of or repetition of what is already stated as rules or guidelines in the specifications that are references in section 1.4.

 

## 1.4. References

| Specification | Location |
|---|---|
| SOAP 1.1 specification | http://www.w3.org/TR/2000/NOTE-SOAP-20000508/ |
| SOAP 1.2 specification | http://www.w3.org/TR/soap12/ |
| SOAP w/Attachments | http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211 |
| WS-I basic profile | http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html |
| WS-Security spec. | http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss |
| XML-Signature spec. | http://www.w3.org/TR/xmldsig-core/ |
| XML-Encryption spec. | http://www.w3.org/TR/xmlenc-core/ |
| WSDL 1.1 Recommendation | |

## 1.5. Definitions and Conventions

This document provides guidelines regarding the use of the SOAP protocol with the OTA specification. Each guideline is denoted by the symbol '§'. Since this document is recommended guidance and not an enforceable requirement (from the perspective of the OTA), the guidelines herein make use (primarily) of SHOULD nomenclature.

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document are to be interpreted as described in IETF document RFC 2119.

In other words, "MUST," "MUST NOT," "REQUIRED," "SHALL," and "SHALL NOT" identify protocol features that are not optional.

Similarly, "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" identify statements that require consideration with regard to the impact that inclusion or exclusion of the protocol feature will have on the interoperability of the system.

 

# 2. OTA Transport Protocol Reference: SOAP

## 2.1. Philosophy of Interoperability

For maximum interoperability, your system SHOULD be liberal in what it accepts and strict in what it emits.

## 2.2. SOAP version 1.1 and 1.2

SOAP version 1.1 was released as a "Note" from W3C in 2000. This specification had few changes from what was submitted to W3C's XML Protocol Working Group the same year. The main addition was a fixed namespace (http://schemas.xmlsoap.org/soap/envelope/).

The main change in SOAP version 1.2, released as a W3C "Recommendation" in June 2003, was a change in the namespace (now http://www.w3.org/2003/05/soap-envelope) and a change in how the SOAP Action value was transmitted over HTTP.

```
Accept-Language: en
Content-Type: type="text/xml";
Content-Length: 194
SOAPAction: GetStockQuote
User-Agent: Acme SOAP Client

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <getStockQuote Ticker="SUNW"/>
 </soap:Body>
</soap:Envelope>
```

**Figure 7: SOAP 1.1 HTTP Sample**

```
Accept-Language: en
Content-Type: type="text/xml"; action="GetStockQuote";
Content-Length: 193
User-Agent: Acme SOAP Client

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
 <soap:Body>
   <getStockQuote Ticker="SUNW"/>
 </soap:Body>
</soap:Envelope>
```

**Figure 8: SOAP 1.2 HTTP Sample**

Most SOAP toolkits/stacks support both versions of SOAP. By using a standard development kit or SOAP stack one ensures that an application will support both SOAP versions.

 

§ 1. OTA implementers using the SOAP protocol MUST support either SOAP 1.1 *or* SOAP 1.2 for clients and services.

§ 2. OTA implementers using the SOAP protocol SHOULD support SOAP 1.1 *and* SOAP 1.2 for clients and services.

## 2.3. SOAP Messaging and SOAP RPC

### 2.3.1. Differences Between SOAP Messaging and SOAP RPC

The SOAP protocol can be used both for messaging (exchanging messages/documents) and for RPC (invoking remote services). The structure of the SOAP Envelope, Header and Body elements are the same for both paradigms, but the content of the SOAP Body element is different.

When using SOAP messaging for document exchange the content of the SOAP Body element is a document or message that is sent from one party to another. This document or message is the only content inside the SOAP Body, and the root element of the document or message is the only immediate child element of the SOAP Body (i.e. you cannot transmit more than one document inside a single SOAP Body element). The exchanged document or message is often defined using an XML Schema which is either agreed between the two SOAP parties or defined by a standards body (such as OTA).

When using SOAP RPC the content of the SOAP Body element describes a service invocation (a remote procedure call). This description normally includes a procedure or function name and a set of parameters. The parameters can be one or more XML documents or messages, so it is possible to exchange documents and messages using SOAP RPC. The structure of the SOAP Body contents is defined on a per-service basis, and is most often described in a WSDL document that is made available by the SOAP RPC service.

### 2.3.2. WSDL's Document/Literal Binding

WSDL can be used to describe both RPC and document exchange services. For document exchange services the document/literal binding must be used (described in the W3C WSDL 1.1 specification sections 3.3 through 3.5):

 

```
<wsdl:definitions ...>
  ...
  <wsdl:binding name="mySoapBinding" type="impl:SOAPClientInterface">
    <!-- Use document style and not rpc. -->
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

    <!-- Define OTA_Cancel operation using OTA-defined messages. -->
    <wsdl:operation name="OTA_Cancel">
      <wsdlsoap:operation soapAction="OTA_Cancel"/>
      <wsdl:input name="OTA_CancelRQ">
        <wsdlsoap:body namespace="http://www.acme.com/ota" use="literal"/>
      </wsdl:input>
      <wsdl:output name="OTA_CancelRS">
        <wsdlsoap:body namespace=" http://www.acme.com/ota " use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  ...
</wsdl:definitions>
```

**Figure 9: WSDL Document/Literal Binding Example**

### 2.3.3. SOAP Messaging Benefits

The main benefit from using SOAP messaging for exchange of OTA is that the content of the SOAP Body element is always a single OTA message. This means that a SOAP client can use the exact same message structure for any OTA service that uses SOAP messaging. If using SOAP RPC, each OTA service would define its own SOAP message structure (procedure name, etc.), and a client would have to use a different message structure for each SOAP RPC service.

As an example, we can consider a SOAP client that wants to issue an OTA_LowFareSearchRQ request to a set of OTA services; A, B and C. If using SOAP messaging, the client could construct a single SOAP envelope containing an OTA_LowFareSearchRQ and issue it to services A, B and C. If the services were using SOAP RPC, the client would have to first determine the required SOAP RPC message structure for services A, B and C, create one SOAP request for each service (containing the OTA_LowFareSearchRQ as a parameter) and issue the request to each service.

### 2.3.4. SOAP Messaging vs. RPC Guidelines

The overhead of using RPC is obvious, and even more so when considering the similar complexity in handling SOAP RPC response messages. But, the main advantage to using SOAP messaging is ease of integration and interoperation.

 

§ 3. OTA implementers SHOULD use SOAP messaging for transmitting OTA documents between a SOAP client and a SOAP service.

## 2.4. SOAPAction URI

### 2.4.1. SOAP Messaging and the SOAP Action URI

The SOAP Action is a URI that is transmitted with a SOAP Envelope, apart from the envelope's XML content. This URI is meant to carry the *intent* of the SOAP message. With SOAP messaging the SOAP operation (service name) is not included in the SOAP message. The SOAP Action can be used to send an operation name from the client to the service in cases where the operation is not implied by the type of document that is being sent.

### 2.4.2. SOAP Intermediaries

The SOAP Action URI can also be very valuable to SOAP intermediaries, such as a SOAP gateway, router or proxy. Intermediaries can forward or otherwise handle a SOAP message based on its SOAP Action value, without having to parse the SOAP Envelope.

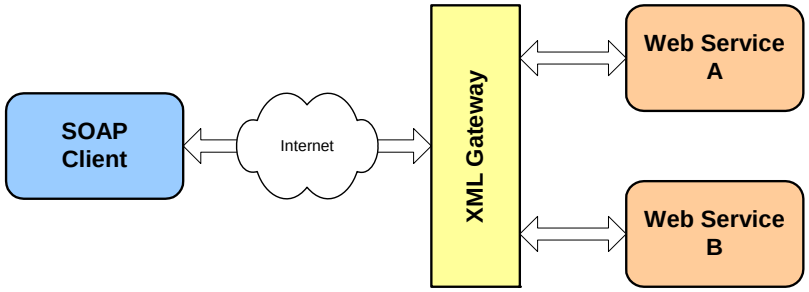


**Figure 10: SOAP Intermediary**

Tasks carried out by such intermediaries include:

- Routing of SOAP requests – routing based on SOAP Action value.
- XML Schema validation of SOAP request contents – XML Schema is chosen from the SOAP Action.
- XML Security token processing, such as XML Signature validation and decryption of XML Encryption elements.

### 2.4.3. SOAP Action URI Guidelines

In scenarios where there is no SOAP intermediary there is most often no need for a SOAP Action URI. The server application can process an incoming message purely based on the root tag of the document it receives.

 

§ 4. Services SHOULD NOT require a SOAP Action URI to communicate with the service.

In scenarios where SOAP intermediaries are used, a SOAP Action URI may be used for in-transit processing. A SOAP server application may also choose to use the SOAP Action URI for internal routing (at an application level).

§ 5. Services and intermediaries that require a SOAP Action URI, SHOULD use the OTA request tag root as the SOAP Action URI. For low fares search, the SOAP Action URI would be “OTA_LowFareSearchRQ”.

Some SOAP intermediaries may not allow an administrator to configure the SOAP Action URI. For this reason a SOAP Client application should allow the SOAP Action URI to be configured per service:

§ 6. Clients SHOULD support any SOAP Action URI for transmitting any OTA message to a Web service.

## 2.5. SOAP Envelope Content

The structure of the SOAP Envelope is well defined in the SOAP 1.1, SOAP 1.2 and WS-I basic profile specifications. WS-I basic profile section 4.1 contains a very detailed description of a properly defined SOAP Envelope. This description does not add, remove or change any of the requirements in SOAP 1.1 or 1.2, but offers the reader a better understanding of the SOAP Envelope structure and constraints than the SOAP documents.

§ 7. SOAP Envelope content SHOULD conform to the SOAP 1.1 and SOAP 1.2 specifications, as clarified in WS-I basic profile section 4.1.

## 2.6. SOAP Header Content

The SOAP Header element is used to carry information that is logically separate from the SOAP payload (contents of the SOAP Body element) but that are required for the SOAP message to reach its endpoint. Such information includes:

- Security tokens
- Routing information
- Timestamps

This element is not intended to carry all or portions of a SOAP message’s payload.

 

§ 8. SOAP Header elements in SOAP request and response messages SHOULD conform to the SOAP 1.1 and SOAP 1.2 specifications, as clarified in WS-I basic profile section 4.1.

§ 9. SOAP Header elements SHOULD NOT contain OTA data.

## 2.7. SOAP Body Content

As SOAP messaging is recommended, the content of the SOAP Body element should be a single OTA request or response message. The only immediate child element of the SOAP Body should be the root element of an OTA message.

§ 10. SOAP Body elements in SOAP request and response messages SHOULD conform to the SOAP 1.1 and SOAP 1.2 specifications, as clarified in WS-I basic profile section 4.1.

§ 11. All content within the SOAP Body element SHOULD be in the OTA namespace, with the exception of XML-Encryption tokens which are inserted in-place of encrypted OTA elements.

§ 12. The content within a SOAP Body element SHOULD be valid, well-formed XML that conforms to an OTA schema.

§ 13. The only immediate child element of the SOAP Body element SHOULD be the root element of a document that is defined in an OTA schema.

## 2.8. SOAP Attachments

SOAP attachments are used to carry information that is in addition to the message payload.

§ 14. SOAP clients SHOULD support SOAP Attachments.

§ 15. SOAP services SHOULD limit the use of SOAP Attachments to images, such as hotel and vehicle images.

## 2.9. SOAP Fault vs. OTA Error

SOAP 1.1 and 1.2 define the SOAP Fault element as a vehicle for transporting error information from a SOAP service to a SOAP client. OTA also defines elements for returning errors and warnings to a SOAP client. While the SOAP Fault element is capable of indicating application-level errors as well as errors that originate in the SOAP stack, this document recommends that the SOAP Fault element be used only to communicate transport (SOAP stack) errors.

 

§ 16. OTA SOAP services SHOULD use SOAP Fault for SOAP-level errors.

§ 17. OTA SOAP services SHOULD use OTA Errors for application-level errors.

§ 18. OTA SOAP clients SHOULD support both SOAP Fault and OTA Error handling.

§ 19. SOAP Fault response messages SHOULD conform to the SOAP 1.1 or SOAP 1.2 specifications, as clarified in WS-I basic profile section 4.1.

 

# 3. Examples

Most of the following examples use the OTA_ReadRQ and OTA_ReadRS messages. The messages are relatively small and thus suitable for short and readable examples.

## 3.1. OTA_ReadRQ - Correct

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <ota:OTA_ReadRQ xmlns:ota="http://www.opentravel.org/OTA/2003/05">
     <ota:POS>
       <ota:Source ISOCountry="CA" ISOCurrency="CAD" PseudoCityCode="AERO">
         <ota:RequestorID ID="APP.001" URL="http://www.acme.com"/>
       </ota:Source>
     </ota:POS>
     <ota:UniqueID ID="105789143" Type="21"/>
   </ota:OTA_ReadRQ>
 </soap:Body>
</soap:Envelope>
```

**Figure 11: Correct OTA_ReadRQ Message**

This shows a correctly formatted SOAP request containing an OTA_ReadRQ message. The OTA_ReadRQ element is the only immediate child element of the SOAP Body element. This format allows the request to be issued to any Web service that implements the OTA_ReadRQ message as a non-RPC service.

## 3.2. OTA_ReadRQ - Incorrect

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <acme:otaReadService xmlns:acme="http://www.acme.com/ota" Request="&lt;ota:OTA_ReadRQ
xmlns:ota=&quot;http://www.opentravel.org/OTA/2003/05&quot;&gt;&lt;ota:POS&gt;&lt;ota:Source
ISOCountry=&quot;CA&quot; ISOCurrency=&quot;CAD&quot;
PseudoCityCode=&quot;AERO&quot;&gt;&lt;ota:RequestorID ID=&quot;APP.001&quot;
URL=&quot;http://www.acme.com&quot;/&gt;&lt;/ota:Source&gt;&lt;/ota:POS&gt;&lt;ota:UniqueID
ID=&quot;105789143&quot; Type=&quot;21&quot;/&gt;&lt;/ota:OTA_ReadRQ&gt;"/>
 </soap:Body>
</soap:Envelope>
```

**Figure 12: Incorrect OTA_ReadRQ Message – Escaped XML**

This is an RPC-type SOAP request containing a call to an “otaReadService” defined by one particular Web Service. This OTA request is serialised and passed to this service as a string. This SOAP message is tied to this particular service and is not in accordance with the guidelines in this document.

 

## 3.3. OTA_ReadRQ - Incorrect

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <acme:otaReadService xmlns:acme="http://www.acme.com/ota">
     <ota:OTA_ReadRQ xmlns:ota="http://www.opentravel.org/OTA/2003/05">
       <ota:POS>
         <ota:Source ISOCountry="CA" ISOCurrency="CAD" PseudoCityCode="AERO">
           <ota:RequestorID ID="APP.001" URL="http://www.acme.com"/>
         </ota:Source>
       </ota:POS>
       <ota:UniqueID ID="105789143" Type="21"/>
     </ota:OTA_ReadRQ>
   </acme:otaReadService>
 </soap:Body>
</soap:Envelope>
```

**Figure 13: Incorrect OTA_ReadRQ Message – Wrapped in other XML**

This message is also an RPC-type SOAP request. Even though the OTA request is not serialised it is still tied to a particular service, thus not in accordance with the guidelines in this document.

## 3.4. OTA_ProfileReadRS – Successful, Correct

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <ota:OTA_ProfileReadRS xmlns:ota="http://www.opentravel.org/OTA/2003/05">
     <ota:Success/>
     <ota:Profiles>
       <ota:ProfileInfo>
         <ota:UniqueID ID="111240578" Type="21"/>
         ...
       </ota:ProfileInfo>
     </ota:Profiles>
         </ota:OTA_ProfileReadRS>
 </soap:Body>
</soap:Envelope>
```

**Figure 14: Correct OTA_ProfileReadRS Successful Response**

The same guideline stating that the SOAP Body element should only contain a single OTA message also applies to SOAP response messages. This message is correctly formatted as the immediate child element of the SOAP Body is an OTA response message root element.

 

## 3.5. OTA_ProfileReadRS – Unsuccessful, Correct

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <ota:OTA_ProfileReadRS xmlns:ota="http://www.opentravel.org/OTA/2003/05">
     <ota:Errors>
       <ota:Error Code="282" Status="NotProcessed" Type="10"/>
     </ota:Errors>
   </ota:OTA_ProfileReadRS>
 </soap:Body>
</soap:Envelope>
```

**Figure 15: Correct OTA_ProfileReadRS Unsuccessful Response**

This message carries an application-level error back to the client application. Application level errors should be reported using OTA Error elements.

## 3.6. OTA_ProfileReadRS – Unsuccessful, Correct

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <soap:Fault>
     <faultcode>soap:MustUnderstand</faultcode>
     <faultstring>SOAP Action URI missing</faultstring>
     <faultactor>http://www.acme.com/services/ota</faultactor>
     <detail>...</detail>
   </soap:Fault>
 </soap:Body>
</soap:Envelope>
```

**Figure 16: Correct OTA_ProfileReadRS Unsuccessful Response**

This response message carries a SOAP-level error back to the client. Such errors should not be generated by an OTA server application, but can be generated by an XML intermediary or the OTA application's SOAP stack/API.

 

## 3.7. Sample SOAP Messaging WSDL for OTA

```
<wsdl:definitions
   targetNamespace="http://www.acme.com/ota"
   xmlns:impl="http://www.acme.com/ota"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:ota="http://www.opentravel.org/OTA/2003/05">

 <!-- Import of request and response schemas. -->
 <xsd:import schemaLocation="OTA_ReadRQ.xsd"/>
 <xsd:import schemaLocation="OTA_ProfileReadRS.xsd"/>

 <!-- No additional data types should be required. -->
 <wsdl:types/>

 <!-- Define request and response message. -->
 <wsdl:message name="OTA_ReadRQ">
   <wsdl:part name="OTA_ReadRQ" element="ota:OTA_ReadRQ"/>
 </wsdl:message>
 <wsdl:message name="OTA_ProfileReadRS">
   <wsdl:part name="OTA_ProfileReadRS" element="ota:OTA_ProfileReadRS"/>
 </wsdl:message>

 <!-- Define SOAP interface. -->
 <wsdl:portType name="SOAPClientInterface">
   <wsdl:operation name="OTA_Read">
     <wsdl:input message="impl:OTA_ReadRQ" name="OTA_ReadRQ"/>
     <wsdl:output message="impl:OTA_ProfileReadRS" name="OTA_ProfileReadRS"/>
   </wsdl:operation>
 </wsdl:portType>

 <!-- Define SOAP binding for SPM. -->
 <wsdl:binding name="spmSoapBinding" type="impl:SOAPClientInterface">
   <!-- Use document style and not rpc. -->
   <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
   <!-- Define operation using defined messages. -->
   <wsdl:operation name="OTA_Read">
     <wsdlsoap:operation soapAction="ReadProfile"/>
     <!-- Use 'literal' to include OTA XML as-is. -->
     <wsdl:input name="OTA_ReadRQ">
       <wsdlsoap:body namespace="http://localhost/services/spm/spm" use="literal"/>
     </wsdl:input>
     <wsdl:output name="OTA_ProfileReadRS">
       <wsdlsoap:body namespace="http://localhost/services/spm/spm" use="literal"/>
     </wsdl:output>
   </wsdl:operation>
 </wsdl:binding>

 <!-- Define SOAP interface with previously declared binding. -->
 <wsdl:service name="SOAPClientInterfaceService">
   <wsdl:port binding="impl:spmSoapBinding" name="spm">
     <wsdlsoap:address location="http://www.acme.com/services/ota"/>
   </wsdl:port>
 </wsdl:service>

</wsdl:definitions>
```

**Figure 17: Sample WSDL Definition for Simple OTA Service**

This outlines how an OTA service can be described in a WSDL document. The only data-types used in this document are types that are already defined in OTA schemas. The <binding/> element declares the service as using the document/literal style to pass the OTA-defined messages as-is between the OTA client and service. Please refer to OTA's WSDL Publication document for further guidelines on implementing WSDL definitions for OTA Web services.

## 3.8. SOAP with Attachments Sample

```
Accept-Language: en
Content-Type: multipart/related; type="text/xml"; boundary="----=Multipart_Boundary_=----"
Content-Id: soap-envelope
Content-Length: 8967

-----=Multipart_Boundary_=----
Content-Type: text/xml
Content-Id: soap-envelope

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
   <OTA_HotelAvailRS xmlns="http://www.opentravel.org/OTA/2003/05">
     <Success/>
     <RoomStays>
       <RoomStay xmlns="http://www.opentravel.org/OTA/2003/05">
         ...
         <BasicPropertyInfo BrandCode="DE" ChainCode="DE" ...>
           <VendorMessages>
             <VendorMessage>
               <SubSection SubTitle="HotelImage">
                 <Paragraph>
                   <URL>cid:HotelPool.jpg</URL>
                 </Paragraph>
               </SubSection>
             </VendorMessage>
             ...
           </VendorMessages>
           ...
         </BasicPropertyInfo>
       </RoomStay>
     </RoomStays>
   </OTA_HotelAvailRS>
 </soap:Body>
</soap:Envelope>
-----=Multipart_Boundary_=----
Content-Type: image/jpeg
Content-Id: HotelPool.jpg

 ...binary image data goes here...
-----=Multipart_Boundary_=------
```

**Figure 18: OTA Message with Hotel Image Attachment**

The above example shows how a binary attachment can be returned with an OTA response message. The SOAP Envelope containing the OTA response is the main part of the response message. The OTA message references an attachment using the 'cid:' URL prefix.

 

## 3.9. WS-Security Token Sample

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

 <soap:Header>
   <wsse:Security soap:actor="acme-ota-proxy" xmlns:wsse="...">
     <wsse:UsernameToken wsu:Id="morten" xmlns:wsu="...">
       <wsse:Username>morten</wsse:Username>
       <wsse:Nonce EncodingType="UTF-8">
         7h9x9UYevqMgBUzT2CXAK2oVYYU2J9PeEG0ZQniz8Tk=
       </wsse:Nonce>
       <wsse:Password Type="wsse:PasswordDigest">
         3gDnTTVAd19NqSVA0c9sl1mNJrM=
       </wsse:Password>
       <wsu:Created>2006-01-17T17:55:14Z</wsu:Created>
     </wsse:UsernameToken>
   </wsse:Security>
 </soap:Header>

 <soap:Body>
   <ota:OTA_ReadRQ xmlns:ota="http://www.opentravel.org/OTA/2003/05">
     <ota:POS>
       <ota:Source ISOCountry="CA" ISOCurrency="CAD" PseudoCityCode="AERO">
         <ota:RequestorID ID="APP.001" URL="http://www.acme.com"/>
       </ota:Source>
     </ota:POS>
     <ota:UniqueID ID="105789143" Type="21"/>
   </ota:OTA_ReadRQ>
 </soap:Body>

</soap:Envelope>
```

**Figure 19: OTA SOAP Message with WS-Security Token**

This sample OTA request contains a WS-Security username/password token. The SOAP Body contains only the OTA request payload, while the security token is placed in the SOAP Header. The security token is typically processed by a SOAP proxy, such as an XML Security Gateway, while the SOAP Body content is processed by an OTA Web service. The security token and OTA request are kept separate, which helps disconnect the OTA Web service from the security layer.

This example illustrates how standard WS-Security compliant implementations may be leveraged by OTA Message producers and consumers. As with all referenced specifications, the goal is to leverage these industry standards. Please refer to the WS-Security specification for further details on this authentication method.

 

## 3.10. XML-Signature Sample

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

 <soap:Header>
   <wsse:Security soap:actor="acme-ota-proxy" xmlns:wsse="...">
     <dsig:Signature Id="MortenJorgensen" xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
       <dsig:SignedInfo>
         <dsig:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         <dsig:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <dsig:Reference URI="">
           <dsig:Transforms>
             <dsig:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
               <dsig:XPath xmlns:ota="http://www.opentravel.org/OTA/2003/05">
                 ancestor-or-self::ota:UniqueID
               </dsig:XPath>
             </dsig:Transform>
             <dsig:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
           </dsig:Transforms>
           <dsig:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
           <dsig:DigestValue>C33iD3LMNL0/4aSey+DdzH76Cng=</dsig:DigestValue>
         </dsig:Reference>
       </dsig:SignedInfo>
       <dsig:SignatureValue>
         nIYgIKl7D0pUKBGPhN36g/vnlNu38fRJ0uooKktbpFWsPQj8A75AXAJo4TYrssgrkJoWZOFnY8h0
         d4PDAJcbgRjFusZYRhlQ3MWEMJ/9GFnLyglNTcveskWNweuUgyz56ARHHnb6MUvEzykMV4So6zaX
         6I8I130T3/r0NaD0DLE=
       </dsig:SignatureValue>
       <dsig:KeyInfo>
         <dsig:KeyName>CN=Morten Jorgensen,O=OpenJaw,...,C=IE</dsig:KeyName>
       </dsig:KeyInfo>
     </dsig:Signature>
   </wsse:Security>
 </soap:Header>

 <soap:Body>
   <ota:OTA_ReadRQ xmlns:ota="http://www.opentravel.org/OTA/2003/05">
     <ota:POS>
       <ota:Source ISOCountry="CA" ISOCurrency="CAD" PseudoCityCode="AERO">
         <ota:RequestorID ID="APP.001" URL="http://www.acme.com"/>
       </ota:Source>
     </ota:POS>
     <ota:UniqueID ID="105789143" Type="21"/>
   </ota:OTA_ReadRQ>
 </soap:Body>

</soap:Envelope>
```

**Figure 20: Signed OTA SOAP Message**

This sample OTA request has an XML Signature element, which contains a digital signature over the OTA payload. Again, the SOAP Body contains only the OTA request payload, while the security token is placed in the SOAP Header. The security token is typically processed by a SOAP proxy, such as an XML Security Gateway, while the SOAP Body content is processed by an OTA Web service. The security token and OTA request are kept separate, which helps disconnect the OTA Web service from the security layer.

Please refer to the XML-Signature and WS-Security specifications for further details.

 

## 3.11. XML-Encryption Sample - Correct

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

 <soap:Header>
   <wsse:Security soap:actor="acme-ota-proxy" xmlns:wsse="...">
     <enc:EncryptedKey xmlns:enc="..." Encoding="utf-8" MimeType="text/xml">
       <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5">
         <enc:KeySize>256</enc:KeySize>
       </enc:EncryptionMethod>
       <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
         <dsig:KeyName> CN=Morten Jorgensen,O=OpenJaw,...,C=IE</dsig:KeyName>
       </dsig:KeyInfo>
       <enc:CipherData>
         <enc:CipherValue>
           I6kwMqpERPpbYoKa2lc/7g4kOuT/ntbpkXbRTX0VQFs6NYOcGItuvpB9qrCe4XKb
           FKsePNOMQmUQQjSzZvIfzhLDst01NZaHCVRZ8FBAGiAWTH06ZPZEDc10WObcy+Y6
           hWJ4Gd0XpDTmaUc3pRei18iWZOYz9no6PjyfJDThIQI=
         </enc:CipherValue>
       </enc:CipherData>
       <enc:CarriedKeyName>session-key</enc:CarriedKeyName>
     </enc:EncryptedKey>
   </wsse:Security>
 </soap:Header>

 <soap:Body>
   <ota:OTA_ReadRQ xmlns:ota="http://www.opentravel.org/OTA/2003/05">
     <ota:POS>
       <ota:Source ISOCountry="CA" ISOCurrency="CAD" PseudoCityCode="AERO">
         <ota:RequestorID ID="APP.001" URL="http://www.acme.com"/>
       </ota:Source>
     </ota:POS>
     <enc:EncryptedData xmlns:enc="..." Encoding="utf-8" MimeType="text/xml"
                        Type="http://www.w3.org/2001/04/xmlenc#Element">
       <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc">
         <enc:KeySize>256</enc:KeySize>
       </enc:EncryptionMethod>
       <dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
         <dsig:KeyName>session-key</dsig:KeyName>
       </dsig:KeyInfo>
       <enc:CipherData>
         <enc:CipherValue>
           fcU3IOeBS/wPKxJgPimX/apgGMnfKjF/hYOceAVLz6avzEn97xV/ipBlMLDTilef
           q4NJWQiwJyqWzXjCQFVEp+OTFav29m6NMc336psfZbpBXLXjNTGBtslVYZ0G2l4P
           FKLFua4nljlr5ve8vxn/AA==
         </enc:CipherValue>
       </enc:CipherData>
     </enc:EncryptedData>
   </ota:OTA_ReadRQ>
 </soap:Body>

</soap:Envelope>
```

**Figure 21: Encrypted OTA Sample SOAP Message**

XML-Encryption tokens are the only type of security token which is inserted in-place of the SOAP payload (in this case an OTA Request). XML-Encryption tokens are typically used when the message travels across one or more SOAP intermediaries or is processed by multiple Web service that need access to only certain parts of the request.

 

Exchange of encrypted XML normally follows these steps:

- A SOAP client typically creates the request as clear-text XML. The sensitive portions of the request are encrypted either by the SOAP client itself or by a proxy that resides inside the local network of the SOAP client. The SOAP request is encrypted using the public key of the Web service it is intended for.
- The SOAP request can then safely travel across a public network.
- The SOAP request is typically decrypted by an XML Security Gateway before it is forwarded to the Web service. The XML Security Gateway decrypts the SOAP request using the public key of the Web service.

The key point to note is that both the client and service handle OTA messages in clear text, while the encryption proxy and security gateway handle the encryption and decryption processes. This allows the client and server to be implemented as if SOAP messages are transmitted in clear text.

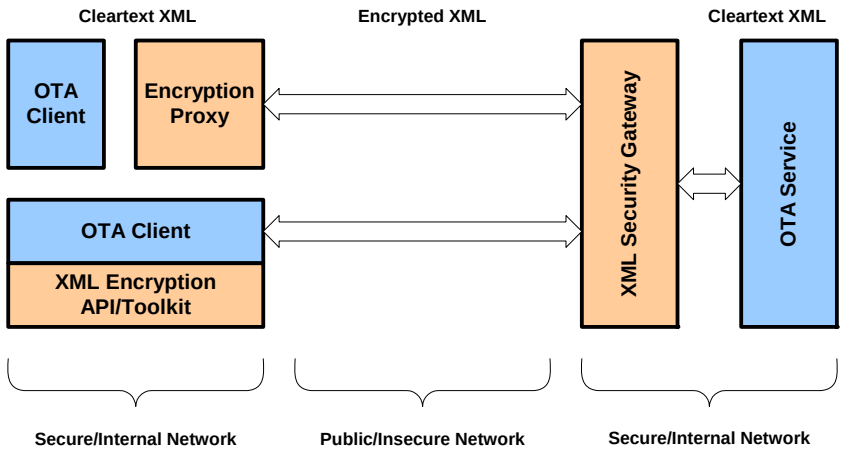


**Figure 22: XML Encryption Scenario**

Please refer to the XML-Encryption specification for further details.