



---

## **Guidelines for Implementing Interoperable Web Services**

Version 1.0 — 28 March 2007

For Comment

---

**National E-Health Transition Authority Ltd**

Level 25

56 Pitt Street

Sydney, NSW, 2000

Australia.

[www.nehta.gov.au](http://www.nehta.gov.au)

**Disclaimer**

NEHTA makes the information and other material ("Information") in this document available in good faith but without any representation or warranty as to its accuracy or completeness. NEHTA cannot accept any responsibility for the consequences of any use of the Information. As the Information is of a general nature only, it is up to any person using or relying on the Information to ensure that it is accurate, complete and suitable for the circumstances of its use.

**Document Control**

This document is maintained in electronic form. The current revision of this document is located on the NEHTA Web site and is uncontrolled in printed form. It is the responsibility of the user to verify that this copy is of the latest revision.

**Copyright © 2007, NEHTA.**

This document contains information which is protected by copyright. All Rights Reserved. No part of this work may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems—without the permission of NEHTA. All copies of this document must include the copyright and other information contained on this page.

# Table of contents

|  |            |
|--|------------|
| <b>Table of contents</b> .....                 | <b>iii</b> |
| <b>Document information</b> .....              | <b>iv</b>  |
| Change history .....                           | iv         |
| <b>1 Introduction</b> .....                    | <b>1</b>   |
| 1.1 Background.....                            | 1          |
| 1.2 Purpose .....                              | 1          |
| 1.3 Scope.....                                 | 1          |
| 1.4 Definitions, acronyms, abbreviations ..... | 3          |
| 1.5 Feedback .....                             | 3          |
| <b>2 Technical Guidelines</b> .....            | <b>4</b>   |
| 2.1 Web service definition .....               | 4          |
| 2.2 Security .....                             | 12         |
| 2.3 Addressing .....                           | 22         |
| <b>Appendix A: References</b> .....            | <b>25</b>  |

# Document information

## Change history

| Version | Date       | Comments            |
|---------|------------|---------------------|
| 1.0     | 2007-03-28 | Release for comment |

---

# 1 Introduction

## 1.1 Background

The National E-Health Transition Authority (NEHTA) has recommended Web services as the technology for application-to-application communications in Australia's e-health environment.

NEHTA has also published a *Web Services Standards Profile* document that identifies the set of Web service standards that should be used in e-health [WSSP2006].

However, those standards by themselves are not sufficient to ensure interoperability. Those standards are very flexible, which allows them to be implemented in potentially incompatible ways. Practical interoperability also needs to factor in how those standards have been implemented in the products and toolkits. This guidelines document has been developed to support the interoperable use of the Web services standards.

The guidelines in this document have been designed to incorporate:

- Compliance with the *Web Service Standards Profile* [WSSP2006];
- Alignment with the *Technical Architecture for Implementing Services* [TAIS2006]; and
- Interoperability between different Web services toolkits in a cross-platform and inter-organisational environment.

These guidelines complement the standards profiles that other organisations have published to improve interoperability of Web services. The best known of these is the *Basic Profile* from the Web Services Interoperability (WS-I) group [BP2006]. However, the guidelines in this document cover material that is beyond what the WS-I Basic Profile provides. It provides specific guidelines on how particular mechanisms are to be used.

## 1.2 Purpose

This document provides guidelines on how to specify services to comply with the Web service standards in an interoperable manner. These guidelines also cover how to implement Web services in an interoperable manner.

These guidelines should be incorporated into the service interface specifications for specific services. The guidelines do not describe any specific service, so they do not attempt to address everything that a service interface specification needs to address.

## 1.3 Scope

The guidelines in this document focus on how to apply the Web service standards. It covers the design and specification of Web service interfaces using WSDL, and its realisation with SOAP messages.

This document does not cover higher level service design. It also does not cover lower level network details. It does not cover how to implement the guidelines using any particular programming language or toolkit.

These guidelines should be followed when specifying service interfaces. As a consequence of incorporating them into service interface specifications, they will also be followed by the implementations of those specifications.

This document is intended for:

- Solution architects, who need to understand the Guidelines, because the features in them can influence the high level design of systems.
- Developers of service interface specifications, who needs to decide which Guidelines to follow, and to incorporate the Guidelines into the interface specification documents.
- Software developers, who can use the Guidelines to help understand the service interface specification they are implementing.

The reader is expected to have a detailed knowledge of Web services at the implementation and protocol level, and of the NEHTA *Web Services Standards Profile* [WSSP2006].

A familiarity with the NEHTA *Technical Architecture for Implementing Services: Concepts and Patterns* is useful for understanding the motivation behind some of the guidelines [TAIS2006].

### **1.3.1 PKI**

These guidelines currently focus on the use of PKI and X.509 certificates for signing and encryption.

Future revisions of these guidelines may support additional security mechanisms. However, the needs of other security mechanisms have not yet been included into these guidelines.

Services that wish to use other security mechanisms (e.g. SAML tokens) may have to deliberately violate some of these current guidelines.

These guidelines also assume that the X.509 certificates contain Subject Key Identifiers.

### **1.3.2 Usage**

The main use of these guidelines will be for developing service interface specifications. The service interface specifications document how a particular service is used. That specification should adopt these guidelines whenever possible.

It is recommended that all the guidelines be adopted, unless there is a clear reason not to. However, the relevance of a particular guideline to a particular service would depend on the particular requirements of that service. Therefore, the developer of the service interface specification needs to decide which guidelines are relevant to their specific service.

Deviations from the guidelines should be well documented and agreed to by all parties involved. Deviations should not be used without carefully considering the impact on interoperability.

### **1.3.3 Change**

These guidelines reflect current best practice. However, best practice changes over time.

These guidelines may be changed in future releases of this document. New guidelines may be added, or existing guidelines may be removed or modified. Those new guidelines may be incompatible with the existing guidelines. Whilst is desirable to maintain compatibility, it cannot be guaranteed.

New guidelines may be issued to reflect:

- Addition of new Web service standards;
- Addition of new NEHTA standards; and
- Changes to Web services toolkit implementations.

## 1.4 Definitions, acronyms, abbreviations

|                       |   |
|-----------------------|---|
| Toolkit               | A library that implements the Web services protocols. Normally a developer will not implement the Web services protocols by writing their own code. They would use an existing Web services library, either provided with their development platform or by a third party component. |
| Service provider      | A program that offers a Web service. It functions as a server.  |
| Service requestor     | A program that invokes the Web service offered by a service provider. It functions as a client.   |
| SOAP                  | A protocol for using XML-based messages to exchange structured data in a distributed computing environment.   |
| SOAP message creator  | A program that creates and sends a SOAP message. A SOAP message creator may be a service requestor or a service provider (since providers would also need to create responses and faults).  |
| SOAP message consumer | A program that receives and processes SOAP messages. A SOAP message consumer may be a service provider or service requestor (since requestors would also need to consume responses and faults).   |
| HTTP                  | HyperText Transfer Protocol   |
| MEP                   | Message Exchange Pattern  |
| NTP                   | Network Time Protocol   |
| PKI                   | Public Key Infrastructure   |
| RPC                   | Remote Procedure Call   |
| TCP                   | Transmission Control Protocol   |
| URL                   | Uniform Resource Locator  |
| URN                   | Uniform Resource Name   |
| UTC                   | Coordinated Universal Time  |
| UUID                  | Universally Unique Identifier   |
| XML                   | Extensible Markup Language  |

### 1.4.1 Terminology

The keywords **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** in this document are to be interpreted as described in IETF's RFC 2119 [RFC2119].

## 1.5 Feedback

Feedback and contributions to this document is being sought, particularly from health and messaging software vendors, public and private health institutions and government health jurisdiction.

The deadline for feedback on version 1.0 is 30 June 2007.

Please send comments to: [securemessaging@nehta.gov.au](mailto:securemessaging@nehta.gov.au)

## 2 Technical Guidelines

### 2.1 Web service definition

#### Guideline TG1: Structuring of WSDL files

##### Issue

WSDL can be used to describe both the abstract service interface and the concrete aspects of a particular service instance.

To enable reuse, the concrete aspects need to be separated from the abstract aspects. This would allow the concrete aspects to be modified without needing to modify the abstract aspects.

##### Obligation

The WSDL SHOULD be separated into one file containing the interface elements: types, message, portType.

And a different file containing the: binding, service.

##### Notes

The file containing the binding and service elements can use the WSDL import mechanism to refer to the interface WSDL.

#### WSDL containing service interface information:

```
<?xml version="1.0"?>
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:tns="http://..."
  targetNamespace="http://..."
  name="...">

  <types>
    ...
  </types>

  <message name="...">
    ...
  </message>
  ...

  <portType name="...">
    ...
  </portType>

</definitions>
```

#### WSDL containing service instance information:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  ...>

  <import namespace="http://..." location="... .wsdl"/>

  <binding ...>
    ...
  </binding>

  <service name="...">
    ...
  </service>
</definitions>
```

## Guideline TG2: Use “document/literal” encoding

### Issue

The SOAP binding for WSDL 1.1 permits different encoding styles [WSDL2001]. Any given Web service operation must adopt one particular style to use.

Some styles are underspecified and are not as well supported by toolkit implementations. Use of those styles can cause integration problems between different toolkits.

### Obligation

Service interface specifications **MUST** use the “document/literal” encoding.

### Notes

In the “document/literal” style, the structure of the SOAP body and the types used are defined by an XML Schema in the types section in the WSDL.

The literal style is recommended because XML Schema is a mechanism that allows a more expressive description of data than the SOAP encoding mechanism. This style also allows XML Schemas that are developed elsewhere to be used in the messages. The literal style is compliant with the widely implemented WS-I Basic Profile 1.1 [BP2006]<sup>1</sup>.

Modern toolkits currently provide better support for the “document/literal” style, over the other possible styles. Some older toolkits only support RPC style operations. However, those older toolkits lack features needed to satisfy other Guidelines too.

The document style is recommended over the RPC style, because the RPC approach is tightly coupled and low level—making it less useful for cross-organisation operations in the e-health environment.

A coarse grain, document oriented approach should be used with Web services. Aim to send fewer messages, and avoid “chatty” interfaces which frequently send messages.

### Example

This example shows the use of the “document/literal” style in a WSDL description of the service interface.

#### WSDL Definition:

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/">
  ...
  <wsdl:binding ...>
    <wsdl:operation name="...">
      <soap:operation style="document"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
      <wsdl:fault name="...">
        <soap:fault name="..." use="literal"/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

<sup>1</sup> WS-I Basic Profile 1.1 specifies the use of SOAP 1.1—instead of SOAP 1.2, which is specified in NEHTA’s *Web Services Standards Profile* [WSSP2006]. However, the WS-I Basic Profile 1.1 has a strong influence on what features toolkits support, including toolkits that support SOAP 1.2. At the time of writing, there is no WS-I profile covering the use of SOAP 1.2.

**Guideline TG3: Follow the wrapped convention****Issue**

The body of SOAP messages can contain arbitrary XML. However, some toolkits expect the body to follow certain conventions. For example, the SOAP body must contain only one XML element, and that element name must uniquely identify the operation. If these conventions are not followed, the toolkits cannot route or process the messages.

**Obligation**

Service interface specifications **MUST** follow the wrapped convention for structuring the body of the SOAP message.

**Notes**

Following the wrapped convention ensures that implementations satisfy the conventions expected by these toolkits.

The wrapped convention is a constraint on how the SOAP body is structured [BUTE2005]. In the wrapped convention, the WSDL definitions have the following characteristics:

- Input and output messages have only one part.
- Each part references an XML Schema element (i.e. not an XML Schema type). This element is called a wrapper element.
- The wrapper element's type is an XML Schema complex type.
- The wrapper element's type has no attributes.
- The local name of the wrapper element in the input message matches the message's operation name.
- The local name of the wrapper element in the output message matches the message's operation name with "Response" appended to it.

See Guideline TG7 for the conventions on the value of the namespace for the wrapper element.

In the above description, a "part" refers to a WSDL part for the SOAP body. WSDL parts are also used to define SOAP headers, but the wrapped convention does not place any constraints on how the SOAP headers are used.

Some toolkits rely on the XML element in the SOAP body to identify the service operation and routing of messages. There are other mechanisms that can (and should) be used for this purpose (i.e. the WS-Addressing action, which is covered by Guideline TG5). However, some earlier toolkits rely on conventions instead of those other mechanisms. Therefore, using the wrapped convention will enable those toolkits to work, without preventing the newer toolkits from using the other mechanisms.

The wrapped convention ensures that the body element contains at most one child element, even though SOAP allows the possibility of multiple child elements. Knowing there is at most one child element can simplify how the body is processed. Having multiple elements in the body is not compliant with the WS-I Basic Profile 1.1 [BP2006], so using the wrapped convention ensures that this criteria is satisfied.

Use of the wrapped convention can also simplify programming with toolkits [MAME2005]. The wrapped convention is akin to defining the RPC style using the document/literal style. Some toolkits recognise this similarity and generate RPC-style method signatures in stub classes, which is easier for developers to program with.

## Example

This example shows the use of a wrapper element in the WSDL document. The SOAP body refers to a complex type, whose name is the operation name.

### WSDL Definition:

```
<wsdl:definitions
  targetNamespace="http://example.org/DSR"
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://example.org/DSR">

  <wsdl:types>
    <xsd:schema ...>
      <xsd:element name="SendDS">
        <xsd:complexType ... />
      </xsd:element>
      <xsd:element name="SendDSResponse">
        <xsd:complexType ... />
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="SendDSInputMessage">
    <wsdl:part name="body" element="tns:SendDS" />
  </wsdl:message>
  <wsdl:message name="SendDSOutputMessage">
    <wsdl:part name="body" element="tns:SendDSResponse" />
  </wsdl:message>

  <wsdl:portType name="...">
    <wsdl:operation name="SendDS">
      <wsdl:input message="tns:SendDSInputMessage" />
      <wsdl:output message="tns:SendDSOutputMessage" />
    </wsdl:operation>
  </wsdl:portType>
  ...
</wsdl:definitions>
```

### Guideline TG4: Use the SOAP Request-Response message exchange pattern

#### Issue

SOAP allows for a range of different message exchange patterns (MEPs) [SOAP2002]. All Web service operation must adopt one particular MEP to use, although different operations can use different MEPs.

This guideline favours the SOAP Request-Response message exchange pattern, because it is the most stable and well supported. Some of the other MEPs are underspecified and are not as well supported by toolkit implementations. Use of those other MEPs would increase integration problems between toolkits.

#### Obligation

Service interface specifications MUST use the SOAP Request-Response message exchange pattern.

Service interface specifications MUST NOT use any of the other message exchange patterns.

#### Notes

The SOAP Request-Response message exchange pattern requires that an input and an output message type be provided for the operation type in the WSDL definition. Even if the request has no parameters, it needs to send a SOAP request message with a body containing just the wrapper element (see Guideline TG3) which contains an empty content model. Service providers must generate a SOAP response for every request they receive.

The SOAP Request-Response message exchange pattern was chosen because it is the most pragmatic option. There are four possible message exchange patterns specified in WSDL 1.1 [WSDL2001]:

- One-Way
- Request-Response
- Solicit-Response
- Notification

The Solicit-Response and Notification message exchange patterns are not recommended because they are not compliant with WS-I Basic Profile 1.1 [BP2006] and are not well supported by toolkits.

Although the One-Way message exchange pattern is compliant with WS-I Basic Profile 1.1 [BP2006], there are interoperability issues with toolkits implementing one-way messaging over HTTP. For example, when toolkits return a zero-length HTTP response, they set the HTTP headers in different ways. Some toolkits return a MIME type of *application/soap+xml*, while some toolkits do not set the MIME type and the Web server container sets it to a default value like *text/plain*. Some toolkits set the *Content-Length* header to zero, while some do not set *Content-Length* causing the Web server to use HTTP's chunked transfer encoding (which some recipients cannot handle). These variations make integration more difficult when dealing with the One-Way message exchange pattern. Using the SOAP Request-Response MEP avoids these problems.

In addition to improving integration, the use of SOAP Request-Response over One-Way MEP can help ensure that some error conditions are detected. These are usually transport layer errors, which would otherwise be silently discarded if a One-Way MEP was used.

The discussion so far has been about message level exchange patterns. However, there are also application level interactions to consider. These application level interactions are described in the Technical Architecture for Implementing Services [TAIS2006], and are different from the message exchange patterns described here (even though they have similar names). The application level interactions are concerned with the abstract semantics of the operation; the message exchange patterns described here are concerned with the concrete messages that get transmitted. This Guideline recommends the use of SOAP Request-Response MEP, regardless of what application level interaction is being performed.

A SOAP Request-Response MEP must be used, even when only a one-way application interaction is needed. In those situations, a dummy response message must be generated. The dummy SOAP response message's body would contain a wrapper element (see Guideline TG3) that has an empty content model. The service requestor can simply discard the dummy response message. The transmission of a dummy response message seems inefficient, but since HTTP 1.1 is used as the transport mechanism a HTTP response is always generated—so the additional overheads of a dummy response message are not very high.

The Request-Response application level interaction must only be used for short lived operations. Waiting for an operation, which takes a long time to respond, consumes resources and increases the chances of failure. When dealing with services that cross organisational boundaries, latencies, response times, and robustness cannot be guaranteed. The designer of the service interface specification should use two separate one-way application level interactions to implement long lived operations. That is, the service requestor sends/receives one SOAP Request-Response pair of messages to request the operation, and later the service provider sends/receives another SOAP Request-Response pair of messages to return the result.

## Example

This example shows the use of the SOAP Request-Response MEP for every operation. Every operation has both an input and an output message.

Even though the SendDischargeSummary operation has one-way semantics, it is implemented as an operation that has a (dummy) output message.

### WSDL Definition:

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:tns="http://example.org/ns/2007/DischargeSummaryReceiver/v1"
  targetNamespace="http://example.org/ns/2007/DischargeSummaryReceiver/v1"
  name="DischargeSummaryReceiver">
  ...

  <!-- tns:SendDischargeSummaryOutMsg is a dummy message -->

  <portType name="DischargeSummaryReceiver">
    <operation name="Ping">
      <input message="tns:PingInMsg" wsaw:Action="..." />
      <output message="tns:PingOutMsg" wsaw:Action="..." />
    </operation>

    <operation name="SendDischargeSummary">
      <input message="tns:SendDischargeSummaryInMsg" wsaw:Action="..." />
      <output message="tns:SendDischargeSummaryOutMsg" wsaw:Action="..." />
      <fault name="InvalidIdFault" message="tns:InvalidIdFault" wsaw:Action="..." />
    </operation>

    <operation name="CheckStatus">
      <input message="tns:CheckStatusInMsg" wsaw:Action="..." />
      <output message="tns:CheckStatusOutMsg" wsaw:Action="..." />
      <fault name="InvalidIdFault" message="tns:InvalidIdFault" wsaw:Action="..." />
    </operation>
    ...
  </portType>
  ...
</definitions>
```

### Guideline TG5: Provide WS-Addressing Action values for WSDL message types

#### Issue

Web services need a mechanism for identifying the operation.

#### Obligation

Service interface specifications MUST provide WS-Addressing Action values for all messages (i.e. input, output and faults) of an operation.

#### Notes

This Action property is defined as a part of the WS-Addressing specification [WSA2006]. The use of WS-Addressing in WSDL documents is described in [WAWB2006]<sup>2</sup>.

The WS-Addressing binding for WSDL [WAWB2006] defines default values for WS-Addressing Actions. However, this guideline recommends providing them explicitly in the WSDL to avoid any possible problems or confusion.

<sup>2</sup> When this guideline was written, the specification of WS-Addressing in WSDL was a W3C Candidate Recommendation. The specification may change before it becomes a W3C Recommendation, in which case this guideline may be revised.

Guideline TG14 recommends that SOAP messages provide a WS-Addressing Action value. Implementing this guideline requires that WSDL definitions specify WS-Addressing Action values for SOAP messages.

This guideline is the preferred mechanism for identifying operations. However, implementations should also follow Guideline TG6 (which says to provide a SOAP Action value) to ensure compatibility with older toolkits which do may not handle WS-Addressing.

### Example

This example shows the inclusion of WS-Addressing Actions for all input, output, and faults.

#### WSDL Definition:

```
<wsdl:definitions
  targetNamespace="http://example.org/DSR"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wSaw="http://www.w3.org/2006/05/addressing/wsdl">
  ...

  <wsdl:portType name="DSRPort">
    <wsdl:operation name="SendDS">
      <wsdl:input message="..."
        wSaw:Action="http://example.org/DSR/DSRPort/SendDSRequest"/>
      <wsdl:output message="..."
        wSaw:Action="http://example.org/DSR/DSRPort/SendDSResponse"/>
      <wsdl:fault name="BadId" message="..."
        wSaw:Action="http://example.org/DSR/DSRPort/SendDS/Fault/BadId"/>
    </wsdl:operation>
    ...
  </wsdl:portType>
</wsdl:definitions>
```

### Guideline TG6: Provide SOAP Action values for WSDL operation types

#### Issue

Web services need a mechanism for identifying the operation. Some Web services toolkits do not understand WS-Addressing Actions, and must rely upon the older SOAP Action mechanism to identify the operation.

#### Obligation

Service interface specifications **MUST** provide a SOAP Action value for all operation types.

SOAP message consumers **SHOULD** be able to process messages that either do or do not contain the SOAP Action value in the Content-Type HTTP header.

#### Notes

This guideline helps mitigate interoperability issues with toolkits that do not support WS-Addressing, and instead need to use SOAP Action values to route messages. It is expected that WS-Addressing will be used when possible (see Guideline TG5). However, this guideline is provided to improve interoperability with toolkits that must rely upon the older SOAP Action mechanism.

Some Web services toolkits expect the SOAP Action value to be transmitted in the Content-Type HTTP header—even though the SOAP specification says this is optional [SOAP2002]. However, only some toolkits generate this header. Those toolkits rely on other mechanisms to determine the operation (e.g. the WS-Addressing Action or the wrapped convention). Therefore, although this guideline ensures that there will be a SOAP Action in the WSDL, it does not

guarantee that it will appear in the SOAP message. SOAP message consumers must not assume that there will always be a HTTP header containing the SOAP Action.

This guideline ensures that the HTTP header is generated when the toolkits are able to do so. However, it also helps ensure that SOAP message consumers can function with or without the SOAP Action value in the HTTP header.

### Example

This example shows the inclusion of a SOAP Action in the definition of the operation.

#### WSDL Definition:

```
<wsdl:definitions
  targetNamespace="http://example.org/DSR"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wSaw="http://www.w3.org/2006/05/addressing/wsdl">
  ...
  <binding name="DSRBinding" type="tns:DSRPort">
    ...
    <operation name="SendDS">
      <soap:operation style="document"
        soapAction="http://example.org/DSR/DSRPort/SendDSRequest"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
      <fault name="BadId">
        <soap:fault name="BadId" use="literal"/>
      </fault>
    </operation>
    ...
  </wsdl:portType>
</wsdl:definitions>
```

### Guideline TG7: Use consistent Action and namespace values

#### Issue

A number of different identifiers are used in WSDL documents and SOAP messages, and some of these are used for the same or related purposes.

To reduce confusion, for both human readers and computer programs, a consistent scheme for these identifiers should be used.

#### Obligation

Service interface specifications **MUST** use a consistent set of values for the WS-Addressing Action and related values.

The convention to use starts by creating a unique namespace to identify the service, which will be called the "service namespace."

The target namespace for the WSDL description **MUST** be:

```
{service namespace}
```

The WS-Addressing Action value for *input* messages **MUST** be:

```
{service namespace} / {port type name} / {operation name} Request
```

The WS-Addressing Action value for *output* messages **MUST** be:

```
{service namespace} / {port type name} / {operation name} Response
```

The WS-Addressing Action value for *fault* messages MUST be:

{service namespace} / {port type name} / {operation name} / Fault / {fault name}

The SOAP Action value for an operation MUST be:

{service namespace} / {port type name} / {operation name} Request

The XML namespace for the wrapper element MUST be:

{service namespace}

## Notes

These conventions are based on the default convention described in WS-Addressing binding for WSDL [WAWB2006].

## 2.2 Security

### Guideline TG8: Include a WS-Security timestamp

#### Issue

Although the WS-Security specification states that a WS-Security timestamp is optional, it is sometimes needed by toolkits to detect replay attacks.

In some toolkits, replay attack detection is enabled by default. It can be difficult to disable this behaviour so programmers are likely to leave it enabled, causing SOAP messages without a WS-Security timestamp to be rejected.

The accuracy and usefulness of the timestamps will depend on the accuracy of the sending and receiving computer's clocks. Clocks may be inaccurate because of normal clock drift, inaccurate settings, or malicious action. Without external mechanisms to guarantee accuracy, the value of the timestamp cannot be fully depended upon to be correct.

#### Obligation

SOAP message creators MUST include a WS-Security timestamp element.

The timestamp element MUST contain a "Created" time expressed in UTC.

In general, SOAP message consumers MAY ignore any "Expires" timestamp. Therefore SOAP message creators MUST NOT depend upon them being processed.

However, if a specific service interface chooses to use the "Expires" timestamp, it MUST clearly specify the behaviour and the obligations associated with it.

#### Notes

Timestamps are optional in WS-Security [WSS2006]. However, some Web service toolkits require timestamps to be present in the messages they receive (especially the creation time). Therefore, to ensure compatibility with those toolkits, timestamps need to be included in all messages.

The created timestamp will be a time that relates to the transportation of the SOAP message. It should not be used to represent times related to the business process of the message: those timestamps should be included in the body of the SOAP message.

This guideline does not stipulate how the timestamp is used—or if it should be used at all. In fact, some toolkits ignore the timestamp, even if it is provided. Programs must not assume that any timestamps will be processed by the SOAP message consumers.

Since the computer clocks of the SOAP message creator, SOAP message consumer, or both, may be inaccurate, caution should be exercised when using timestamp values. Developers need to be aware of what guarantees are available and what assumptions are being made. For example, using timestamps for detecting replay attacks assumes that the sender's clock does not repeat itself (because the clock has been adjusted backwards, or multiple messages appear to be generated at the same time due to the limited precision of the timestamp). Use the timestamps, but be aware of their limitations.

The WS-Security timestamp element may contain an "Expiry" timestamp element. However, toolkits cannot be depended upon to honour it. Therefore, it should not be relied upon as the only mechanism for dealing with out-of-date messages. With some toolkits it is difficult to remove the Expires timestamp from the SOAP messages they generate—this is acceptable, as long as no expectation is placed on whether it will be processed or not by the SOAP message consumer. In general, it can be used, but don't rely upon it. If a specific service does depend on it, then its behaviour must be explicitly defined as a part of the service interface specification.

Clock accuracy can be addressed using trusted time servers and clock synchronisation protocols such as Network Time Protocol (NTP). Although highly recommended, it is outside the scope of this document to mandate the use of NTP. Even if time synchronisation is mandated, it does not eliminate the possibility of malicious parties from falsifying time values and does not change how the toolkits process expiry times.

### Example

This example shows the WS-Addressing creation timestamp in the SOAP message.

#### SOAP Message:

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
  <soap:Header>

    <wsse:Security>
      <wsu:Timestamp>
        <wsu:Created>2007-01-26T02:55:38Z</wsu:Created>
      </wsu:Timestamp>
      ...
    </wsse:Security>

  </soap:Header>
  <soap:Body> ... </soap:Body>
</soap:Envelope>
```

**Guideline TG9: Sign SOAP messages****Issue**

The integrity of the SOAP message needs to be determined by SOAP message consumers. The SOAP message consumer needs to be able to verify that it has not been tampered with.

The identity of the SOAP message sender needs to be authenticated by SOAP message consumers. The SOAP message consumer needs to be sure of who sent it.

**Obligation**

SOAP message creators **MUST** sign the SOAP body and headers.

SOAP message creators **MAY** sign the SOAP body and headers in any order.

SOAP message consumers **SHOULD** verify that the signature is correct and matches the SOAP body and headers.

SOAP message consumers **SHOULD NOT** rely on the data in any part of the message which has not been signed.

SOAP message consumers **MUST NOT** assume that items are signed in any particular order.

See Guideline TG12 for the signing algorithms.

**Notes**

Digital signatures can be used to verify the integrity of the data and to authenticate the sender. The Web Services Standards Profile [WSSP2006] recommends the use of the WS-Security standard for representing the digital signature in the SOAP message.

This guideline applies to all SOAP messages: requests, responses, and faults.

The signature should be created by the initial sender of the SOAP message.

This guideline only covers the transportation of the message, and not signatures relating to business functionality. If signatures are required relating to business functionality of the message content, then they should be incorporated into the body of the message. The signature on the SOAP message only relates to identifying the entity that sent the message.

All data in the message must be signed, because any unsigned data could be tampered with and poses a security risk. This means the SOAP body and all headers are signed. The only exception is the security header, in which only the timestamp portion must be signed.

This guideline states that the entire SOAP body is signed. Although the XML Signature mechanism allows signing only specific parts of an XML document, this is not used here because the purpose of this signature is to ensure the integrity of the entire SOAP body.

Different toolkits may generate the signature block in a different order. For example, some may sign the body before the WS-Security timestamp, and some the other way around. Therefore it is important for SOAP message consumers to handle signatures that sign its components in any order.

All digital signatures must be calculated over the entire element (i.e. including the start and end tags of the element, not just the contents of the element). This is equivalent to setting the "entire header and body signatures" property in WS-SecurityPolicy to true [WSPL2005].

**Guideline TG10: Encrypt SOAP messages****Issue**

The confidentiality of the data in the SOAP message needs to be preserved, so that third parties cannot access the data.

**Obligation**

SOAP message creators **MUST** encrypt the parts of the SOAP message that need to be kept confidential. See Guideline TG12 for the encryption algorithms.

The entire SOAP body **MUST** be encrypted.

The message signature header **MUST** be encrypted. However, all other headers **SHOULD NOT** be encrypted.

**Notes**

This guideline only applies to SOAP messages that contain confidential data, and that data can be encrypted as a whole. This guideline does not need to be used when there is no confidential data.

NEHTA's *Web Services Standards Profile* [WSSP2006] recommends the use of the WS-Security standard, which uses XML Encryption, for representing the encrypted data in the SOAP message.

The entire SOAP body is encrypted as a single block. Although XML Encryption allows portions of the XML document to be encrypted, this feature is not used here. Specifying that the entire body is encrypted is simpler to understand and verify—reducing the risk that sensitive data is not encrypted. The level of support for partial encryption in toolkits has not been tested. Currently, there are no use cases which require encryption of parts of the body. However, if new use cases for partial encryption emerge, this guideline may be modified to allow for partial encryption.

The headers (other than the header containing the message signature) usually contains data that is used by the toolkits or intermediaries, so they should not be encrypted otherwise those parties would not be able to access them.

The message signature is encrypted, because this can slightly reduce the risk of the message being tampered with. However, this risk is very small: so if the body does not need to be encrypted, don't bother about encrypting the signature.

It should be noted that the correct way to apply XML Encryption to a SOAP message is to use:

- Element encryption on the signature header. This ensures that the element start and end tags are also encrypted.
- Element content encryption on the SOAP body. This ensures that the SOAP body start and end tags are not encrypted.

Toolkits can be very strict about which elements are encrypted and which are not. This is why this guideline suggests that other headers should not be encrypted. If additional headers need to be encrypted, it must be well documented as a part of the service interface. In the future, this guideline may be modified to include other headers in the set that must be encrypted.

**Guideline TG11: Sign before encrypting the SOAP messages****Issue**

The order in which signing and encryption occurs matters. When messages are encrypted before being signed, the signature is computed over the cipher text. When messages are signed before encrypted, the signature is computed over the plain text.

SOAP messages will be rejected, if the SOAP message creator generates a message that is different from what the SOAP message consumer expects. For example, if a message, that was signed before it was encrypted, was sent to a SOAP message consumer that expected it to be encrypted before it was signed. Therefore a specific order needs to be agreed upon.

**Obligation**

SOAP message creators **MUST** sign messages before encrypting them.

**Notes**

Sign-before-encrypt was chosen because it is more useful.

Sign-before-encrypt has been chosen, because the performance advantages of encrypt-before-sign do not apply in this environment. An argument for choosing encrypt-before-sign is that it allows the message's integrity to be checked first, before wasting processing time on decrypting the data if it has been corrupted. However, there is very little advantage to do this in this Web services environment, because integrity problems will normally not occur. In this environment the transport will be HTTP over TCP, and TCP has its own checksums and retransmission mechanism to ensure integrity. Therefore there is no performance argument for using encrypt-before-sign.

The use of sign-before-encrypt helps prevent man-in-the-middle attacks. For instance, an attacker can listen in on requests from a service requestor. The attacker can copy the encrypted data from a request, create a new SOAP request and sign it with a different key that is still recognised by the service provider. Using encrypt-before-signing, the service provider would decrypt and verify the rogue SOAP request, and treat it as a legitimate request. Although the attacker cannot access the data in the request, it can access the response and also possibly infer what was in the request. Thus, attackers only need to obtain one set of keys in order to compromise the messages for everyone in the system. This attack is not possible with sign-before-encrypting since the signature verification will fail when the service provider tries to process the rogue SOAP request.

The sign-before-encrypt approach may also have benefits in auditing, if the receiver already stores the received data in an unaltered form. It would only need to store the signature in an audit trail, because it corresponds to the data that is already being stored; therefore saving storage space. However, if encrypt-before-sign was used, it would have to store the decrypted data, the signature and the encrypted data in an audit trail.

In the WS-Security header, the position of the timestamp may vary between different toolkits. Some toolkits control the order of signing and encrypting through a mechanism that also determines where the timestamp is placed in the security header. Other toolkits do not allow the position of the timestamp to be changed. Therefore, SOAP message consumers must be able to accept messages regardless of where the timestamp appears. This is equivalent to the Lax layout rules in WS-SecurityPolicy [WSPL2005].

|   |
|---|
| <b>Guideline TG12: Use WS-SecurityPolicy's Basic256 algorithm suite</b> |
|---|

**Issue**

Different cryptographic algorithms can be used in WS-Security. However, if the SOAP message creator uses an algorithm that the SOAP message consumer doesn't understand, then they will be incompatible.

**Obligation**

Use the cryptographic algorithms in the Basic256 algorithm suite from WS-SecurityPolicy 1.1 [WSPL2005]. Namely, SOAP message creators **MUST** use these algorithms:

- Encrypt data using the symmetric AES-256 algorithm;
- Encrypt the symmetric key that was used to encrypt data using the RSA OAEP algorithm;
- Transform data using Exclusive XML Canonicalization before calculating digest values;
- Calculate digest values using the SHA-1 algorithm; and
- Sign digest values using the RSA SHA-1 algorithm.

**Notes**

The RSA public key algorithms (i.e. RSA SHA-1 and RSA OAEP), SHA hashing algorithms (i.e. SHA-1), and AES symmetric encryption algorithms (i.e. AES-256) are approved by the Australian Government Defence Signals Directorate [ACSI33].

This guideline selects an algorithm suite from WS-SecurityPolicy to reduce interoperability issues with toolkits whose security configuration is based on WS-SecurityPolicy. In addition, the algorithms in the WS-SecurityPolicy are generally better supported by most Web service toolkits.

These algorithms are identified by the URLs in this table:

| Algorithm        | Name   | Identifier  |
|------------------|--|---|
| Data encryption  | AES-256  | <a href="http://www.w3.org/2001/04/xmlenc#aes256-cbc">http://www.w3.org/2001/04/xmlenc#aes256-cbc</a>         |
| Key encryption   | RSA OAEP   | <a href="http://www.w3.org/2001/04/xmlenc#rsa-oeap-mgf1p">http://www.w3.org/2001/04/xmlenc#rsa-oeap-mgf1p</a> |
| Digest           | SHA-1  | <a href="http://www.w3.org/2000/09/xmldsig#sha1">http://www.w3.org/2000/09/xmldsig#sha1</a>                   |
| Signature        | RSA SHA-1  | <a href="http://www.w3.org/2000/09/xmldsig#rsa-sha1">http://www.w3.org/2000/09/xmldsig#rsa-sha1</a>           |
| Canonicalization | Exclusive XML                                    | <a href="http://www.w3.org/2001/10/xml-exc-c14n#">http://www.w3.org/2001/10/xml-exc-c14n#</a>                 |
| Transformations  | None (other than Exclusive XML canonicalization) |   |

Note: RSA SHA-1 and the Exclusive XML Canonicalization algorithms come from WS-SecurityPolicy's base algorithm suite, which is used by WS-SecurityPolicy's Basic256 algorithm suite.

**Example**

This example shows a SOAP request message that uses the cryptographic algorithms described in this guideline.

This example shows the signature block in plain text (i.e. not encrypted) so that it can be read. However, in a correct SOAP message that follows Guideline TG10, the signature block would be encrypted.

**SOAP Message:**

```

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <soap:Header>
    <wsse:Security>
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p"/>
        ...
      </xenc:EncryptedKey>
      ...
      <ds:Signature>
        <!-- This element should actually be encrypted -->
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
          <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <ds:Reference ...>
            <ds:Transforms>
              <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            ...
          </ds:Reference>
          ...
        </ds:Signature>
      ...
    </wsse:Security>
  </soap:Header>

  <soap:Body>
    <xenc:EncryptedData ...>
      <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
      ...
    </xenc:EncryptedData>
  </soap:Body>
</soap:Envelope>

```

**Guideline TG13: Use the Direct Reference mechanism for including the service requestor's signing X.509 certificate in the request, and the X.509 certificate's Subject Key Identifier mechanism for all other certificate references**

**Issue**

WS-Security has a number of different approaches for communicated in the SOAP message the certificates being used for signing and encryption. SOAP message consumers and creators need to use compatible approaches to ensure interoperability.

**Obligation**

A service requestor **MUST** include its signing certificate in requests using Direct Reference.

Service requestors **MUST** reference the service provider's encryption certificate using that certificate's Subject Key Identifier in requests.

Service providers **MUST** reference their signing certificate using that certificate's Subject Key Identifier in responses.

Service providers MUST reference the service requestor's encryption certificate using that certificate's Subject Key Identifier in responses.

## Notes

This guideline is strict in the mechanisms used in the requests and responses, because some toolkits are very strict in what they expect to process. This applies especially to toolkits whose security configuration is based on the WS-SecurityPolicy standard [WSPL2005]. For example, if the recipient token inclusion policy is set to "Never," then requests where the encryption certificate is included in the message will be rejected.

The two mechanisms used here are:

- Direct Reference used with a copy of the full X.509 certificate included in the SOAP message. The certificate is placed with the SOAP message, so the recipient does not need to lookup any external certificate directory to obtain it<sup>3</sup>.
- Reference to a Subject Key Identifier. The Subject Key Identifier field from the X.509 certificate is placed in the SOAP message. The recipient will have to obtain a copy of the certificate from an external certificate directory. The Subject Key Identifier is a unique identifier for the X.509 certificate.

The Direct reference and Subject Key Identifier mechanisms have been chosen because they are commonly supported by toolkits. Although WS-Security and the X.509 Certificate Token Profile [XCTP2006] describe other mechanisms, they are not well supported by toolkits.

The Subject Key Identifier mechanism is generally preferred, because in most cases there is no need to include the certificate in the message. One case where the inclusion of certificate has some advantages is having the service requestor's signing certificate in the request message, which is why the Direct reference mechanism is recommended for this certificate. For instance, the service provider can verify a request message's digital signature without having to retrieve the requestor's certificate (assuming they trust the root certificate authority of the certificate). In addition, the service provider can use this certificate to encrypt the response (assuming the requestor's certificate permits both digital signature and encryption). In both cases, the service provider does not have to rely upon an external certificate directory.

SOAP message consumers should verify any certificate before using it. For example, it needs to check if the certificate has not been revoked. Also, the service provider does not have to use the service requestor's certificate that was included in the SOAP message: it is allowed to retrieve it from an alternative trusted source.

The table below summarises this guideline:

|                              | <b>Signature Key</b>   | <b>Encryption Key</b>   |
|------------------------------|--|---|
| <b>Request SOAP message</b>  | Direct reference<br><br>(Service requestor's certificate)                    | Reference the Subject Key Identifier<br><br>(Service provider's certificate)  |
| <b>Response SOAP message</b> | Reference the Subject Key Identifier<br><br>(Service provider's certificate) | Reference the Subject Key Identifier<br><br>(Service requestor's certificate) |

<sup>3</sup> Even though the mechanism is called "Direct Reference" it is not being used to refer to the certificate, but is used to implant or embed the certificate in the SOAP message. However, the word "embedded" is not used here, because it is the name of a different mechanism in WS-Security.

The equivalent of this guideline in WS-SecurityPolicy 1.1 is:

```

AsymmetricBinding/InitiatorToken/IncludeToken =
    http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/AlwaysToRecipient
AsymmetricBinding/InitiatorToken/RequireKeyIdentifierReference =
    True
AsymmetricBinding/RecipientToken/IncludeToken =
    http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512/Never
AsymmetricBinding/RecipientToken/RequireKeyIdentifierReference =
    True

```

## Example

This example shows how certificates are handled in SOAP messages.

This SOAP request message contains (in order):

- Reference to the Subject Key Identifier for the encrypting certificate of the service provider (i.e. the consumer of this SOAP message);
- Direct reference (i.e. implanted copy of) the signing certificate of the service requestor (i.e. the creator of this SOAP message);
- Signature block for the message, which indicates that the signer's certificate is the one implanted in this SOAP message.

This example shows the signature block in plain text (i.e. not encrypted) so that it can be read. However, in a correct SOAP message that follows Guideline TG10, the signature block would be encrypted.

### SOAP Request:

```

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wss="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

  <soap:Header>
    <wss:Security>
      <!-- Service provider's encryption certificate referenced using SKI -->
      <xenc:EncryptedKey>
        ...
      <ds:KeyInfo>
        <wss:SecurityTokenReference>
          <wss:KeyIdentifier
            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509SubjectKeyIdentifier"> ...
          </wss:KeyIdentifier>
        </wss:SecurityTokenReference>
      </ds:KeyInfo>
    </xenc:EncryptedKey>

    <!-- Service requestor's signing certificate included in this message -->
    <wss:BinarySecurityToken
      wsu:Id="123"
      EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
      ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"> ...
    </wss:BinarySecurityToken>

    <!-- Signature block, which references the
      service requestor's signing certificate given above -->
    <ds:Signature>
      <!-- This element should actually be encrypted -->
      ...
    </ds:Signature>
  </wss:Security>
</soap:Header>

```

```

        <wsse:SecurityTokenReference>
          <wsse:Reference
            URI="#123"
            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509v3"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    ...
  </wsse:Security>

</soap:Header>
<soap:Body>
  <!-- This is encrypted data -->
  ...
</soap:Body>
</soap:Envelope>

```

This SOAP response message contains (in order):

- Reference to the Subject Key Identifier for the encrypting certificate of the service requestor (i.e. the consumer of this SOAP message);
- Signature block for the message. Inside this block is a reference to the Subject Key Identifier the signing certificate of the service provider (i.e. the creator of this SOAP message).

### SOAP Response:

```

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
  xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <soap:Header>
    <wsse:Security>

      <!-- Service requestor's encryption certificate referenced using SKI -->
      <xenc:EncryptedKey>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier
              EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary"
              ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509SubjectKeyIdentifier">
              ... </wsse:KeyIdentifier>
            </wsse:SecurityTokenReference>
          </ds:KeyInfo>
        </xenc:EncryptedKey>

      <!-- Signature block, which references the
      Service provider's signing certificate using SKI -->
      <ds:Signature>
        ...
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier
              EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary"
              ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
x509-token-profile-1.0#X509SubjectKeyIdentifier">
              ...
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        </ds:Signature>
      ...
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <!-- This is encrypted data -->
    ...
  </soap:Body>
</soap:Envelope>

```

## 2.3 Addressing

**Guideline TG14: Provide WS-Addressing To, From, Action and MessageID headers in SOAP request messages.**

### Issue

The Web Services Standards Profile [WSSP2006] recommends the use of the WS-Addressing 1.0 specification. However, the WS-Addressing specification does not say when or how its headers should be used. This could create interoperability problems for requests.

### Obligation

Service requestors **MUST** provide the following WS-Addressing headers in request SOAP messages:

- *To*,
- *From*,
- *Action*, and
- *MessageID*.

Service requestors **MAY** include other WS-Addressing headers in requests.

Service requestors **MUST** set the *To* field to a logical identifier of the service provider.

Service requestors **SHOULD** set the *From* field to a logical identifier of itself.

The MessageID field **MUST** contain a UUID formatted as a URN.

### Notes

A logical address is one that is not tied to the physical address of where a particular service instance has been deployed. Normally, a fixed endpoint address is used in these WS-Addressing headers—namely a URL for the Web service endpoint. However, this does not allow the message to be routed through intermediaries. Since the addresses are digitally signed by the original sender, it cannot be changed by the intermediaries to resend the message to the next location.

This problem can be solved by using a logical address, which does not ever need to be changed. The logical address can be translated into a physical end-point address by using a Service Instance Directory, as described in [TAIS2006].

The exact form of the logical address is yet to be determined. However, systems should be designed to handle multiple types of logical addresses. Possible logical address could include X.500 Distinguished Names, OpenID URLs, and Healthcare Provider Identification numbers [OID2007].

The WS-Addressing *To* field allows routing of the message to occur.

The WS-Addressing *From* field is included for auditing purposes, and can be used as the default reply-to address.

The WS-Addressing *Action* field can be used to route the request messages. For example, toolkits can map SOAP messages to operation calls using the Action value. Intermediaries can specify particular routes for messages based on their Action value. The Action values should follow Guideline TG7.

The WS-Addressing *MessageID* field is included to support auditing and to detect duplicate messages. The Universally Unique Identifier (UUID) expressed as a URN is a practical value that is globally unique [RFC4122].

In keeping with Guideline TG9, all of these WS-Addressing headers must be digitally signed.

### Example

This example shows the required WS-Addressing headers in a SOAP request.

#### SOAP Request:

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <soap:Header>
    ...
    <wsa:To>urn:HPI:8000000000000002</wsa:To>
    <wsa:From><wsa:Address>urn:HPI:8000000000000001</wsa:Address></wsa:From>
    <wsa:Action>http://example.org/DSR/DSRPort/SendDSRequest</wsa:Action>
    <wsa:MessageID>urn:uuid:652d329a-cd1e-11db-8314-0800200c9a66</wsa:MessageID>
    ...
  </soap:Header>
  <soap:Body> ... </soap:Body>
</soap:Envelope>
```

### Guideline TG15: Provide WS-Addressing Action, MessageID and RelatesTo headers in SOAP response messages

#### Description

The Web Services Standards Profile [WSSP2006] recommends the use of the WS-Addressing 1.0 specification. However, the WS-Addressing specification does not say when or how its headers should be used. This could create interoperability problems for responses sent from a service provider to a service requestor.

#### Obligation

Service providers **MUST** provide the following WS-Addressing headers in response SOAP messages:

- *Action*,
- *MessageID*, and
- *RelatesTo*.

Service providers **MAY** include other WS-Addressing headers in responses.

#### Notes

The WS-Addressing *Action* field can be used to route the response messages.

The WS-Addressing *MessageID* field is included to support auditing. It is a globally unique identifier for the response message. It can also be used to detect duplicate messages.

The WS-Addressing *RelatesTo* field is included to associate the current response message to the request message that initiated it. Its value should be the *MessageID* of the corresponding request. It is also useful for auditing purposes.

In keeping with Guideline TG9, all of these WS-Addressing headers must be included in the data that is digitally signed.

### Example

This example shows the required WS-Addressing headers in a SOAP response.

**SOAP Response:**

```
<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <soap:Header>
    ...
    <wsa:Action>http://example.org/DSR/DSRPort/SendDSResponse</wsa:Action>
    <wsa:MessageID>urn:uuid:037dfbd0-cd1e-11db-8314-0800200c9a66</wsa:MessageID>
    <wsa:RelatesTo>urn:uuid:652d329a-cd1e-11db-8314-0800200c9a66</wsa:RelatesTo>
    ...
  </soap:Header>
  <soap:Body> ... </soap:Body>
</soap:Envelope>
```

# Appendix A: References

- [ACSI33] Defence Signals Directorate, *Australian Government Information and Communications Technology Security Manual*, ACSI 33, 2005-09-19.
- [BP2006] WS-I, *Basic Profile*, 1.1, 2006,  
<http://www.ws-i.org/Profiles/BasicProfile-1.1.html>
- [BUTE2005] Russell Butek, *Which Style of WSDL Should I Use?*, IBM, 24 May 2005,  
<http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/#N1021F>
- [MAME2005] Anne Manes, *The "wrapped" document/literal convention*, 20 March 2005,  
<http://atmanes.blogspot.com/2005/03/wrapped-documentliteral-convention.html>
- [OID2007] OpenID, <http://openid.net/>
- [RFC2119] IETF, *RFC 2119: Keywords for use in RFCs to Indicate Requirement Levels*, S. Bradner, March 1997,  
<http://ieft.org/rfc/rfc2119.txt>
- [RFC4122] IETF, *RFC 4122: A Universally Unique Identifier (UUID) URN Namespace*, P. Leach, M. Mealling, R. Salz.  
<http://ietf.org/rfc/rfc4122.txt>
- [SOAP2002] W3C, *SOAP*, 1.2, 2003,  
<http://www.w3.org/TR/soap12-part2>
- [TAIS2006] NEHTA, *Technical Architecture for Implementing Services*, version 1.0, 21 December 2006.
- [WAWB2006] W3C, *Web Services Addressing 1.0—WSDL Binding*, W3C Candidate Recommendation, 29 May 2006,  
<http://www.w3.org/TR/2006/CR-ws-addr-wsdl-20060529/>
- [WSA2006] *Web Services Addressing 1.0 – Core*, W3C Recommendation, 9 May 2006,  
<http://www.w3.org/TR/2006/REC-ws-addr-core-20060509>.
- [WSDL2001] *Web Services Description Language (WSDL) 1.1*, W3C Note, 15 March 2001,  
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- [WSPL2005] IBM, Microsoft and Verisign, *Web Services Security Policy Language (WS-SecurityPolicy)*, version 1.1, July 2005,  
<http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf>
- [WSS2006] OASIS, *Web Services Security: SOAP Message Security 1.1*, OASIS Standard, 1 February 2006,  
<http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [WSSP2006] NEHTA, *Web Services Standards Profile*, version 2.0, 2006.
- [XCTP2006] OASIS, *Web Services Security X.509 Certificate Token Profile 1.1*, OASIS Standard Specification, 1 February 2006,  
<http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-x509TokenProfile.pdf>